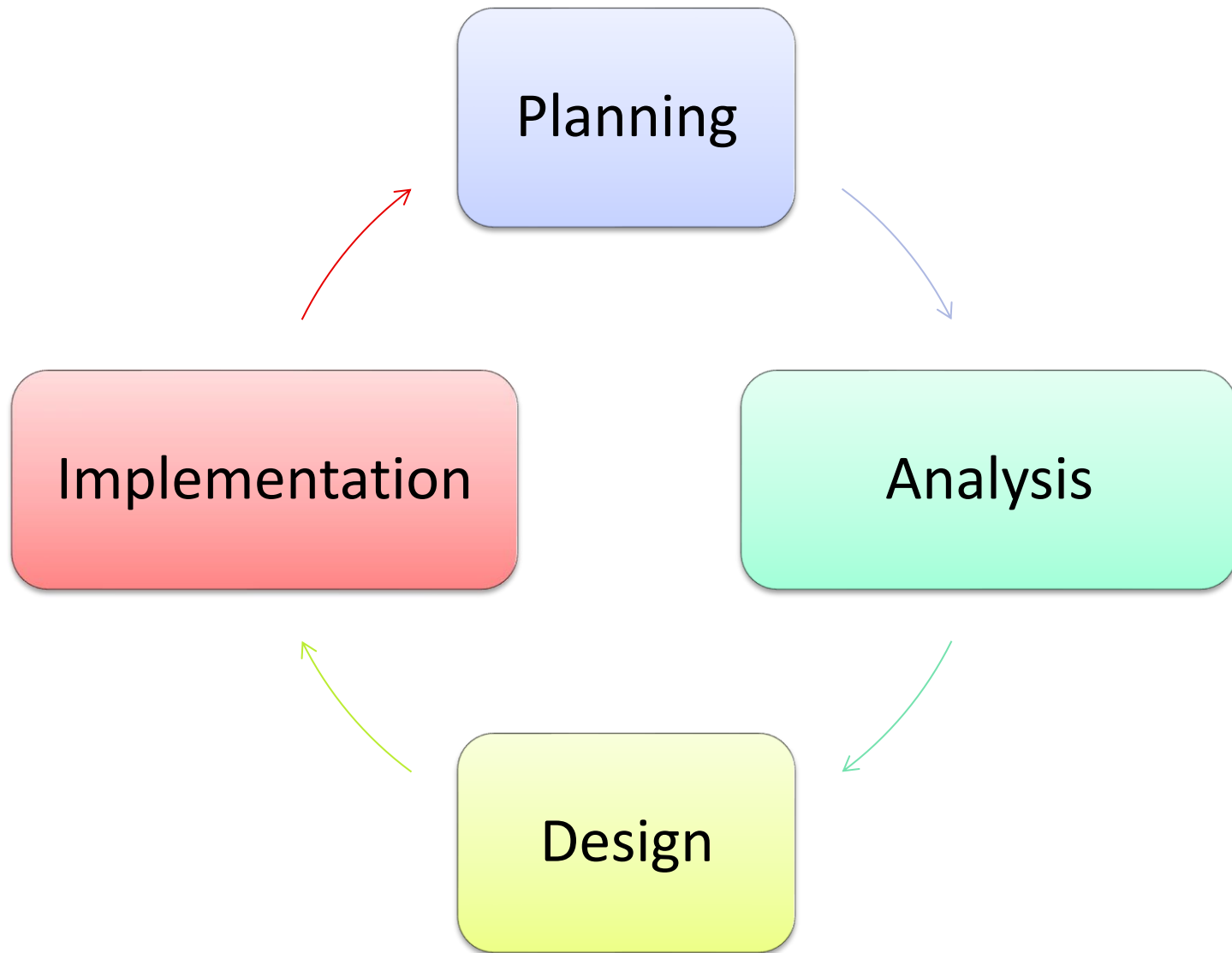


Software Engineering Process

Contents

1. Software Development Life Cycle (SDLC)
2. Software Development Methodologies
3. Software Development Notation and Tools:
 1. Unified Modeling Language (UML) Notation
 2. Computer Aided Software Engineering (CASE) Tools

Software Development Life Cycle (SDLC)



Project Phases

1. **Planning:** Why build the system?

- Identifying Business Value, Feasibility Analysis (System Request/Proposal)

2. **Analysis:** Who, what, when, where will the system be?

- User Requirement Gathering (System Requirements)

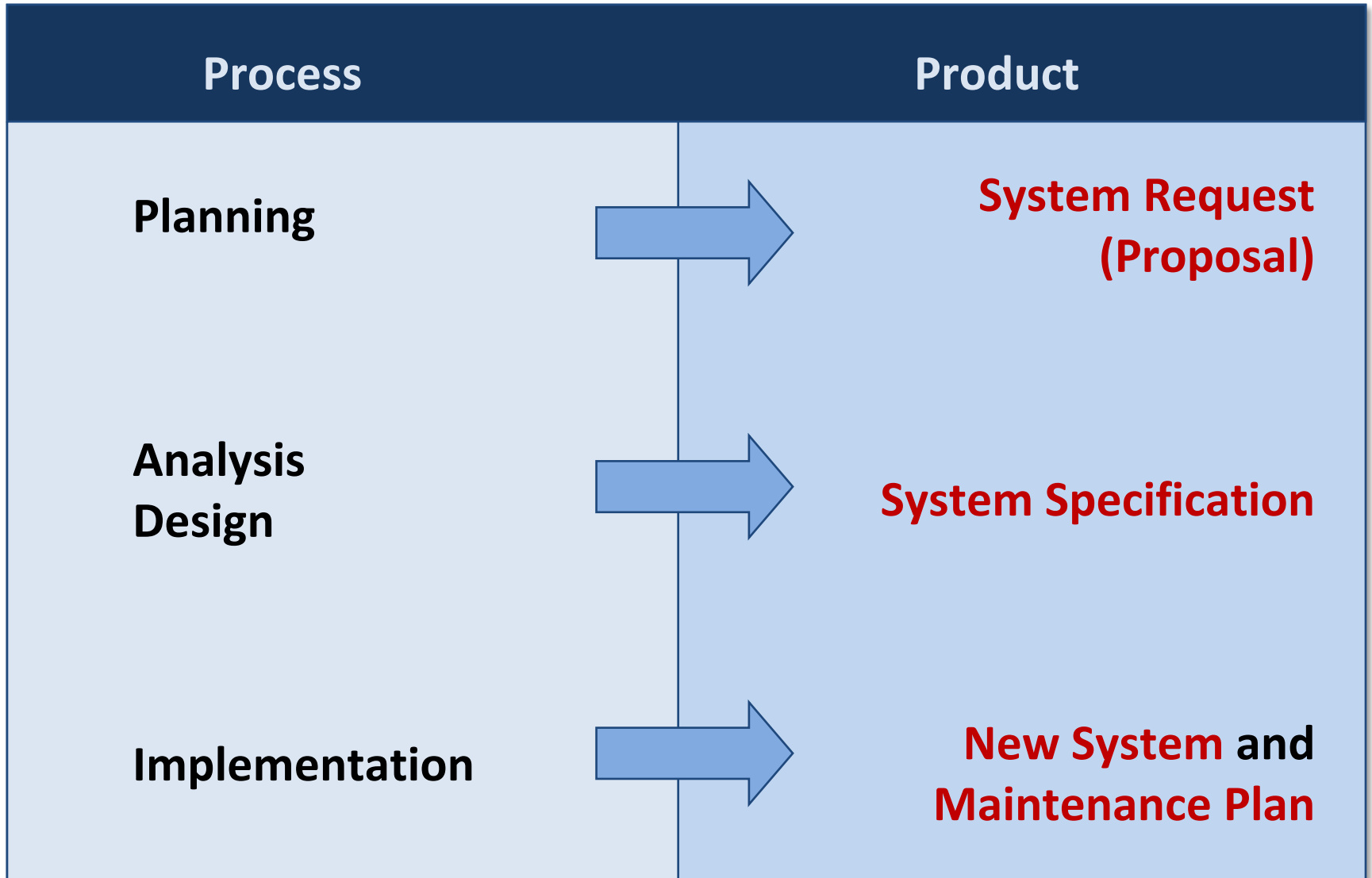
3. **Design:** How will the system work?

- Architecture Design, Interface Design, Data Design, Program Design (System Specification)

4. **Implementation:** System delivery

- System Construction, Testing, Installation, Maintenance (New System)

Processes and Deliverables



Planning

1. Identifying **business value**

- Lower costs
- Increase profits

2. Analyze **feasibility**

- Technical Feasibility
- Economic Feasibility
- Organizational Feasibility

3. Develop **workplan**

4. **Staff** the project : temporer, budget terbatas, waktu juga sudah ada jadwalnya

System Request—Internet order project

Project sponsor:

Margaret Mooney, Vice President of Marketing

Business Need:

This project has been initiated to reach new Internet customers and to better serve existing customers using Internet sales support.

Business Requirements:

Using the Web, customers should be able to search for products and identify the brick-and-mortar stores that have them in stock. They should be able to put items on hold at a store location or place an order for items that are not carried or not in stock. The functionality that the system should have is listed below:

1. Search through the CD Selections' inventory of products
2. Identify the retail stores that have the product in stock
3. Put a product on hold at a retail store and schedule a time to pick up the product
4. Place an order for products not currently in stock or not carried by CD Selections
5. Receive confirmation that an order can be placed and when it will be in stock

Business Value:

We expect that CD Selections will increase sales by reducing lost sales due to out-of-stock or nonstocked items and by reaching out to new customers through its Internet presence. We expect the improved services will reduce customer complaints, primarily because 50 percent of all customer complaints stem from out of stocks or nonstocked items. Also, CD Selections should benefit from improved customer satisfaction and increased brand recognition due to its Internet presence.

Conservative estimates of tangible value to the company includes:

1. \$750,000 in sales from new customers
2. \$1,875,000 in sales from existing customers
3. \$50,000 yearly reduction in customer service calls

Special Issues or Constraints:

The Marketing Department views this as a strategic system. This Internet system will add value to our current business model, and it also will serve as a proof of concept for future Internet endeavors. For example, in the future, CD Selections may want to sell products directly over the Internet.

The system should be in place for the holiday shopping season next year.

Feasibility Analysis Template

Technical Feasibility: Can We Build It?

1. Familiarity with Application: Less familiarity generates more risk
2. Familiarity with Technology: Less familiarity generates more risk
3. Project Size: Large projects have more risk
4. Compatibility: The harder it is to integrate the system with the company's existing technology, the higher the risk

Economic Feasibility: Should We Build It?

1. Development costs
2. Annual operating costs
3. Annual benefits (cost savings and revenues)
4. Intangible costs and benefits

Organizational Feasibility: If We Build It, Will They Come?

1. Project champion(s)
2. Senior management
3. Users
4. Other stakeholders
5. Is the project strategically aligned with the business?

Internet Order Feasibility Analysis Executive Summary

Margaret Mooney and Alec Adams created the following feasibility analysis for the CD Selections Internet Order System Project. The System

Proposal is attached, along with the detailed feasibility study. The highlights of the feasibility analysis are:

Technical Feasibility

The Internet Order System is feasible technically, although there is some risk.

CD Selections' risk regarding familiarity with Internet order applications is high

- The Marketing Department has little experience with Internet-based marketing and sales.
- The IT Department has strong knowledge of the company's existing order systems; however, it has not worked with Web-enabled order systems.
- Hundreds of retailers that have Internet Order applications exist in the marketplace.

CD Selections' risk regarding familiarity with the technology is medium

- The IT Department has relied on external consultants and an Information Service Provider to develop its existing Web environment.
- The IT Department has gradually learned about Web systems by maintaining the current Web site.
- Development tools and products for commercial Web application development are available in the marketplace, although the IT department has little experience with them.
- Consultants are readily available to provide help in this area.

The project size is considered medium risk

- The project team likely will include less than ten people.
- Business user involvement will be required.
- The project timeframe cannot exceed a year because of the Christmas holiday season implementation deadline, and it should be much shorter.

The compatibility with CD Selections' existing technical infrastructure should be good

- The current Order System is a client-server system built using open standards. An interface with the Web should be possible.
- Retail stores already place and maintain orders electronically.
- An Internet infrastructure already is in place at retail stores and at the corporate headquarters.
- The ISP should be able to scale their services to include a new Order System.

Economic Feasibility

A cost–benefit analysis was performed; **see attached spreadsheet for details**. A conservative approach shows that the Internet Order System has a good chance of adding to the bottom line of the company significantly.

- **ROI** over 3 years: 229 percent
- **Total benefit** after three years: \$3.5 million (adjusted for present value)
- **Break-even** occurs: after 1.7 years

Intangible Costs and Benefits

- Improved customer satisfaction
- Greater brand recognition

Organizational Feasibility

From an organizational perspective, this project has low risk. The objective of the system, which is to increase sales, is aligned well with the senior management's goal of increasing sales for the company. The move to the Internet also aligns with Marketing's goal to become more savvy in Internet marketing and sales.

The project has a project champion, Margaret Mooney, Vice President of Marketing. Margaret is well positioned to sponsor this project and to educate the rest of the senior management team when necessary. To date, much of senior management is aware of and supports the initiative.

The users of the system, Internet consumers, are expected to appreciate the benefits of CD Selections' Web presence. And, management in the retail stores should be willing to accept the system, given the possibility of increased sales at the store level.

Additional Comments:

- The Marketing Department **views this as a strategic system**. This Internet system will add value to our current business model, and it also will serve as a proof of concept for future Internet endeavors.
- We should consider **hiring a consultant with expertise in similar applications** to assist with the project.
- We will need to hire **new staff to operate the new system**, from both the technical and business operations aspects.

	2003	2004	2005	Total
Increased sales from new customers	0	750,000	772,500	
Increased sales from existing customers	0	1,875,000	1,931,250	
Reduction in customer complaint calls	0	50,000	50,000	
Total Benefits:	0	2,675,000	2,753,750	
PV of Benefits:	0	2,521,444	2,520,071	5,041,515
PV of All Benefits:	0	2,521,444	5,041,515	
Labor: Analysis, Design and Implementation	162,000	0	0	
Consultant Fees	50,000	0	0	
Office Space and Equipment	7,000	0	0	
Software and Hardware	35,000	0	0	
Total Development Costs:	254,000	0	0	
Labor: Webmaster	85,000	87,550	90,177	
Labor: Network Technician	60,000	61,800	63,654	
Labor: Computer Operations	50,000	51,500	53,045	
Labor: Business Manager	60,000	61,800	63,654	
Labor: Assistant Manager	45,000	46,350	47,741	
Labor: 3 Staff	90,000	92,700	95,481	
Software upgrades and licenses	4,000	1,000	1,000	
Hardware upgrades	5,000	3,000	3,000	
User training	2,000	1,000	1,000	
Communications charges	20,000	20,000	20,000	
Marketing expenses	25,000	25,000	25,000	
Total Operational Costs:	446,000	452,700	464,751	
Total Costs:	700,000	452,700	464,751	
PV of Costs:	679,612	426,713	425,313	1,531,638
PV of all Costs:	679,612	1,106,325	1,531,638	
Total Project Costs Less Benefits:	(700,000)	2,222,300	2,288,999	
Yearly NPV:	(679,612)	2,094,731	2,094,758	3,509,878
Cumulative NPV:	(679,612)	1,415,119	3,509,878	
Return on Investment:	229.16%	(3,509,878/1,531,638)		
Break-even Point:	1.32 years	(break-even occurs in year 2; [2,094,731 – 1,415,119] / 2,094,731 = 0.32)		

Analysis

1. Requirement Gathering by answering the questions:
 - **Who** will use the system?
 - **What** will the system do?
 - **When** will it be used?
2. Investigate the **current system**
3. Identify possible **improvements**
4. Develop a **concept for new system**

Design

1. Architecture design

- Hardware description
- Software description
- Network infrastructure

2. User Interface design

- How users interact with system
- Forms / reports used by the system

3. Data Design

- What data is to be stored
- What format the data will be in
- Where the data will be stored

4. Program Design

- What programs need to be written
- Exactly what each program will do

Design

- These deliverables:
 1. Architecture design (deployment diagram)
 2. User Interface design
 3. Database design (ER diagram)
 4. Program design (UML)

(System Specification)

- The System Specification is **given to the programming team** for implementation

Implementation

1. Construction

- New system is built and tested
- Often testing is the longest part

2. Testing

1. Unit Testing (Blackbox and Whitebox Testing)
2. Integration Testing
3. System Testing
4. User Acceptance Test

3. Installation

- Old system is turned off
- New system is turned on

Software Development Methodologies

What Is a Methodology?

- A formalized **approach to implementing** the SDLC (series of steps and deliverables)
- Writing code **without a well-thought-out** system request may work for small programs, but **rarely works for large ones**

Major Methodologies

1. **Structured Design**

- Waterfall Method
- Parallel Development

2. **Rapid Application Development (RAD)**

- Phased Development
- Prototyping
- Throw-away Prototyping

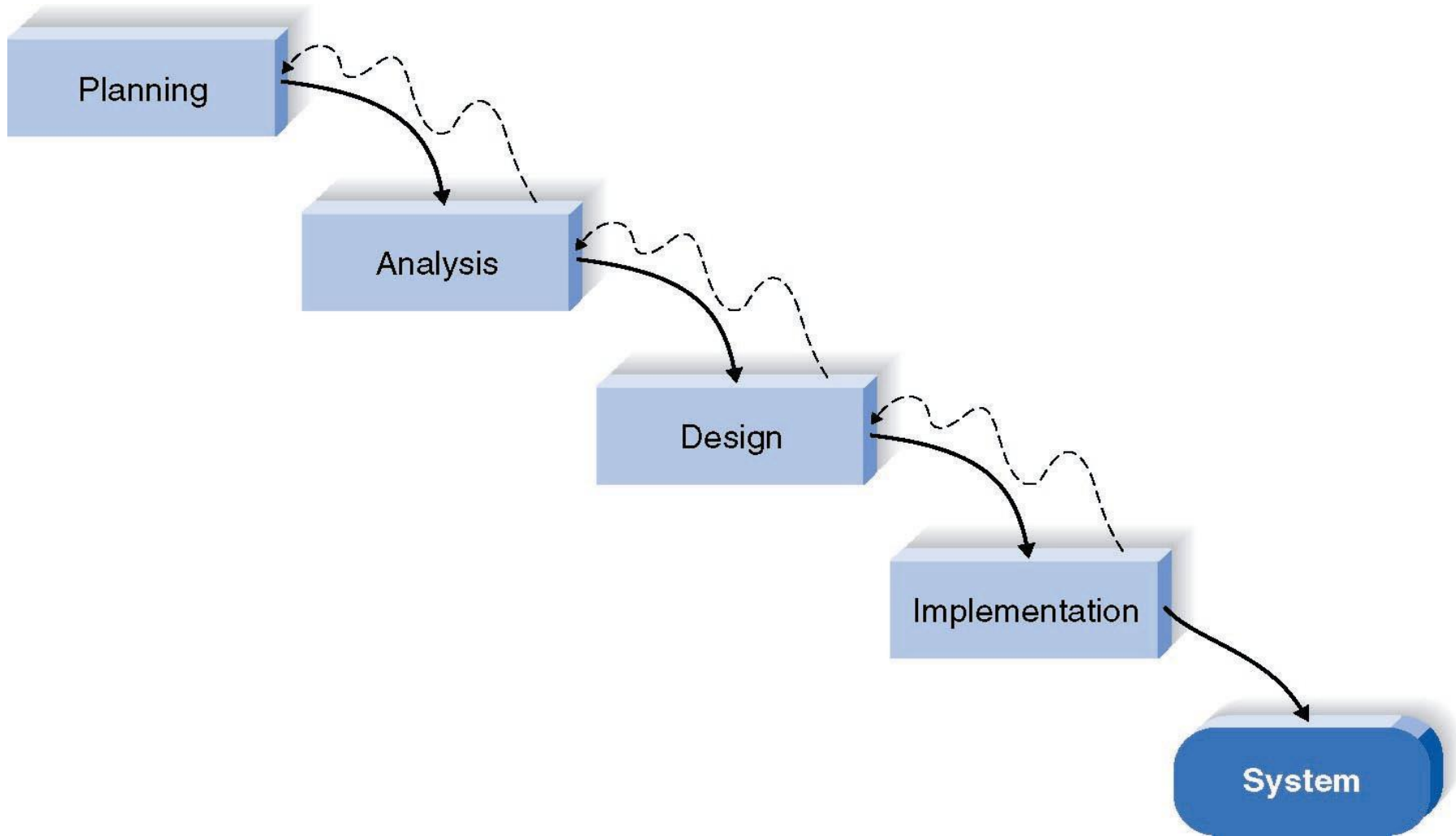
3. **Agile Development**

- Extreme Programming (XP)
- Scrum

Structured Design Methodology

- Projects **move methodically** from one to the next step
- Generally, **a step is finished before the next one begins**

Waterfall Method



Pros - Cons of the Waterfall Method

Pros	Cons
Identifies systems requirements long before programming Begins, it minimizes change to the requirements as the project proceed (mature)	Design must be specified on paper before programming begins
	Long time between system proposal and delivery of new system
	Rework is very hard

Parallel Development

- Addresses problem of time gap between proposal and delivery
- General process:
 1. Breaks project into parallel subproject
 2. Integrates them at the end

Parallel Development

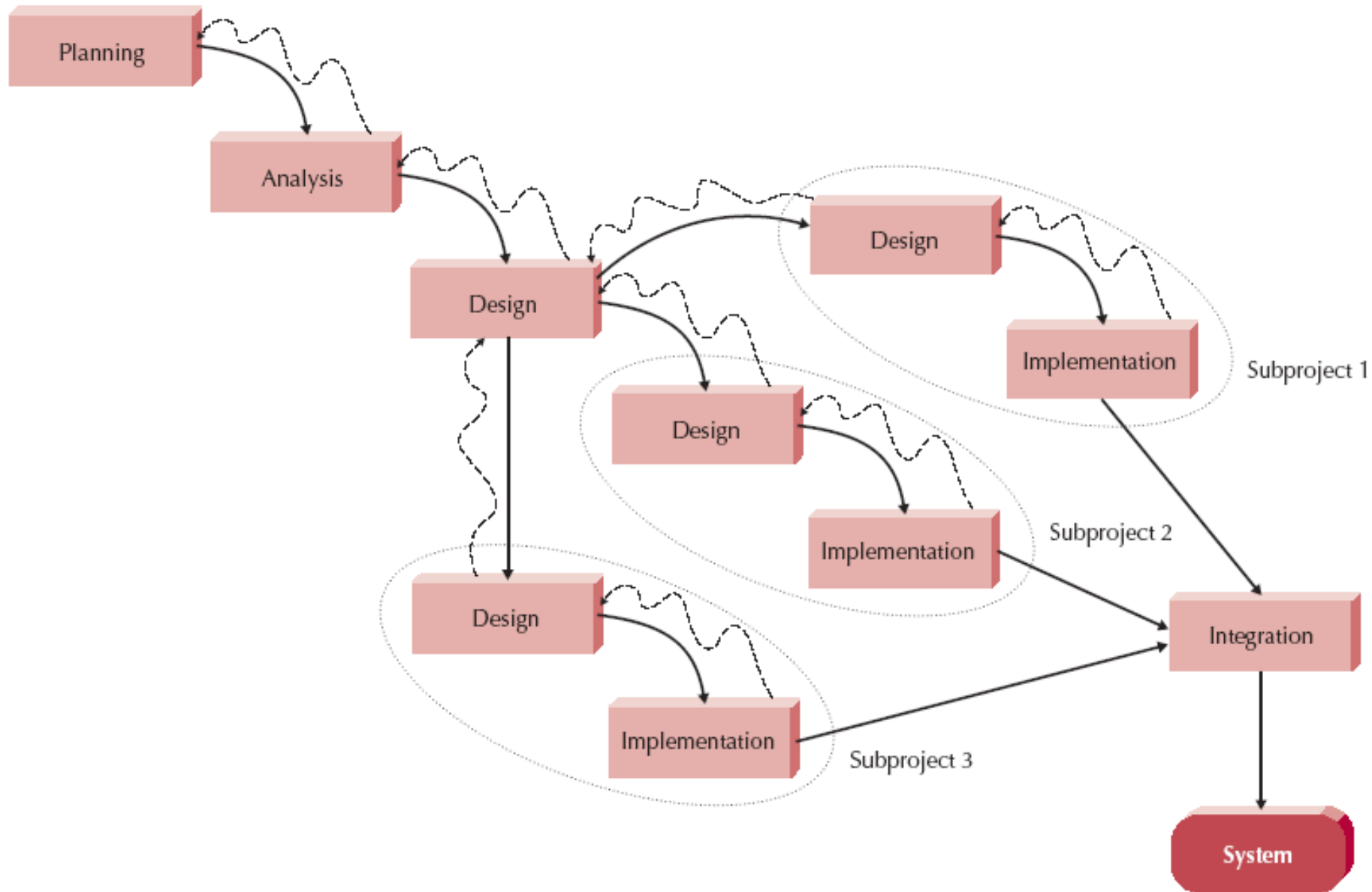


FIGURE 1-3 A Parallel Development-based Methodology

Rapid Application Development

1. Phased development

- A series of versions

2. Prototyping

- System prototyping

3. Throw-away prototyping

- Design prototyping

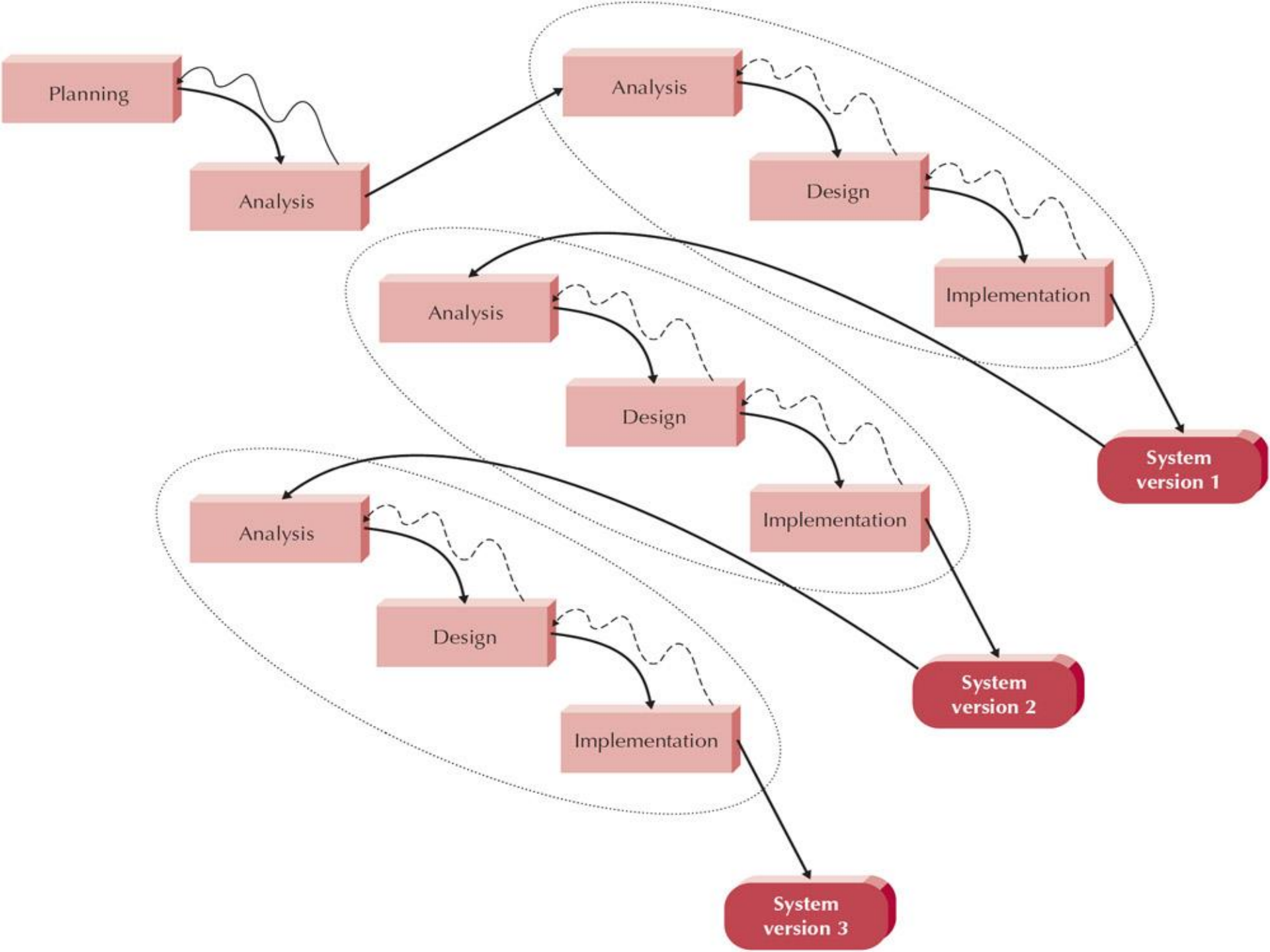
Rapid Application Development

Critical elements to speed up the SDLC:

1. CASE tools
2. Visual programming languages
3. Code generators

RAD: Phased Development

- Break overall system into a series of versions
- Each version has Analysis, Design, and Implementation
- Output from one version is the input to the next
- Incorporate ideas, issues, lessons learned in one version into the next version



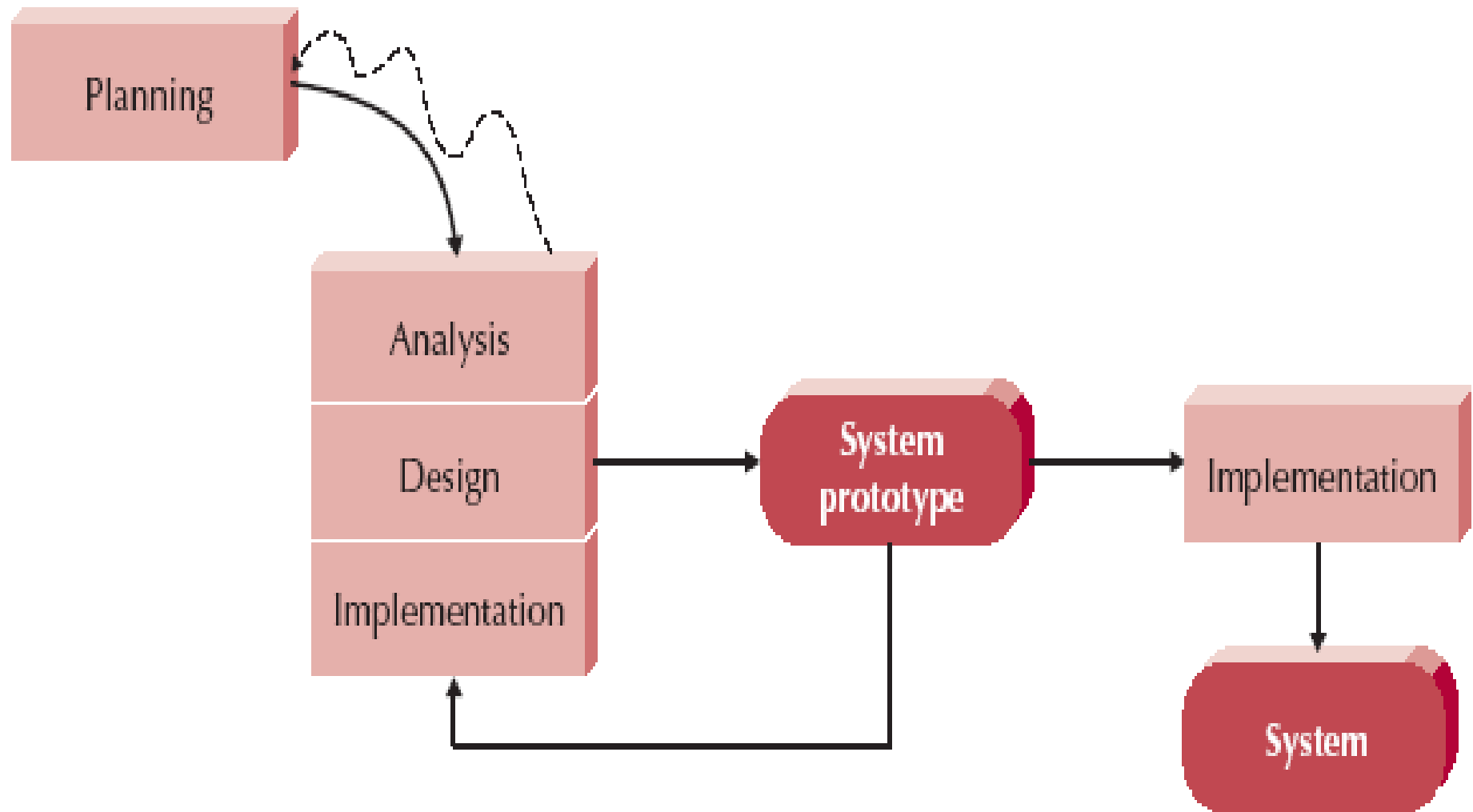
RAD: Phased Development

Pros	Cons
Gets useful system to users quickly Most important functions tested most	Initial system is intentionally incomplete System requirements expand as users see versions

RAD: Prototyping

- Analysis, Design, Implementation are performed concurrently
- Start with a "quick-and-dirty" prototype
 - Provides minimal functionality
- Repeat process, refining the prototype each time
- Stop when prototype is a working system

RAD: Prototyping



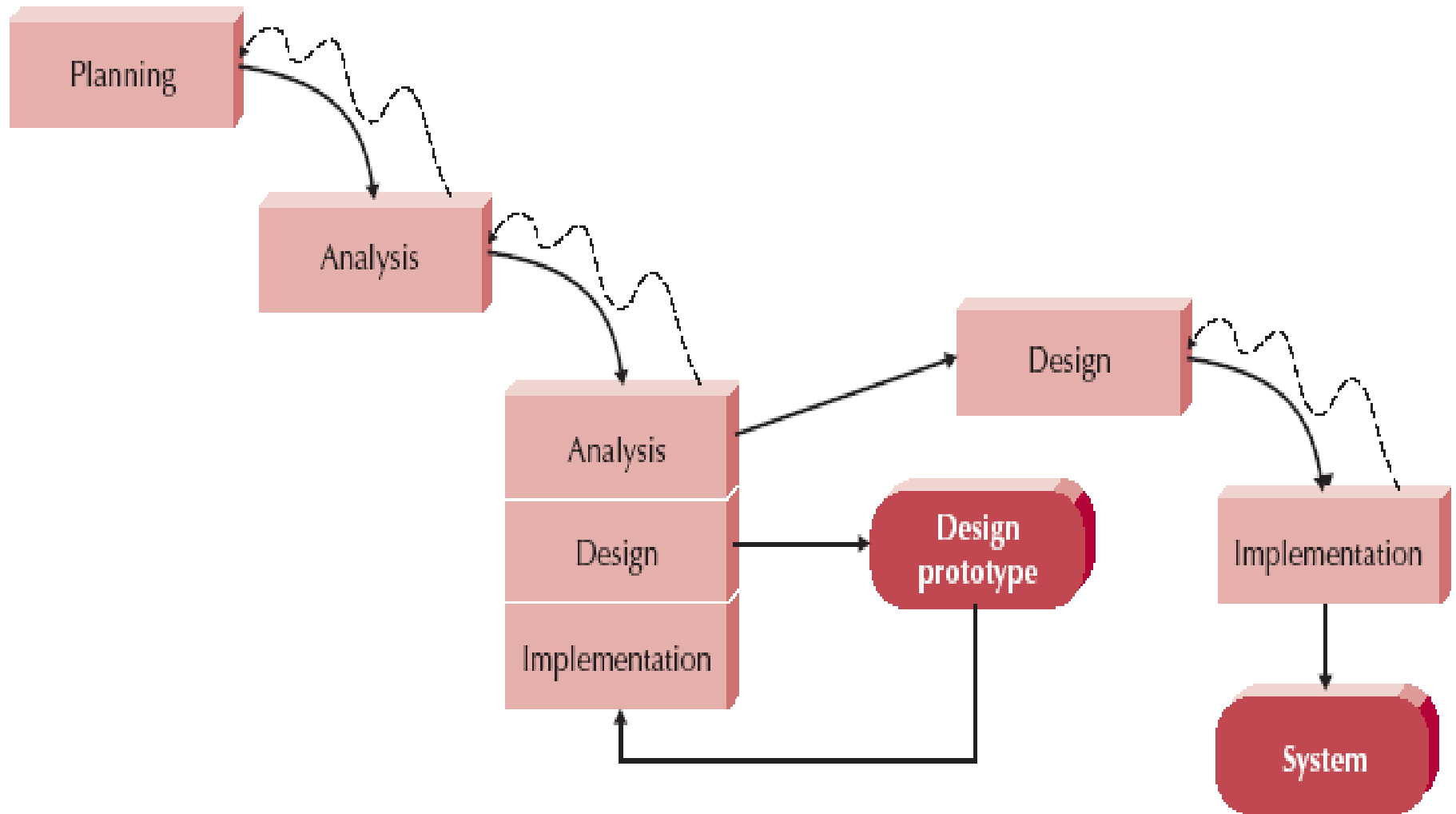
RAD: Prototyping

Pros	Cons
Gets working system to users quickly	Fast paced. Hard to conduct careful, methodical analysis
Reassures users that the project is progressing	Initial design decisions have long term staying power
Quickly refines true requirements	Problems may come to light late in design, requiring re-design

RAD: Throw-Away Prototyping

- Use prototypes only to understand requirements
 - Example: use html to show UI
- Prototype is **not** a working design
- Once requirements are understood, the prototypes are **thrown away**
- The system is then built using SDLC

RAD: Throw-Away Prototyping



Agile Development

- Just a **few rules** that are easy to learn and follow
- Streamline the SDLC
 - **Eliminate** much of the **modeling and documentation**
 - Emphasize **simple, iterative** application development
- Examples include:
 - **Extreme Programming (XP)**
 - **Scrum**
 - Dynamic Systems Development Model (DSDM)

Extreme Programming (XP)

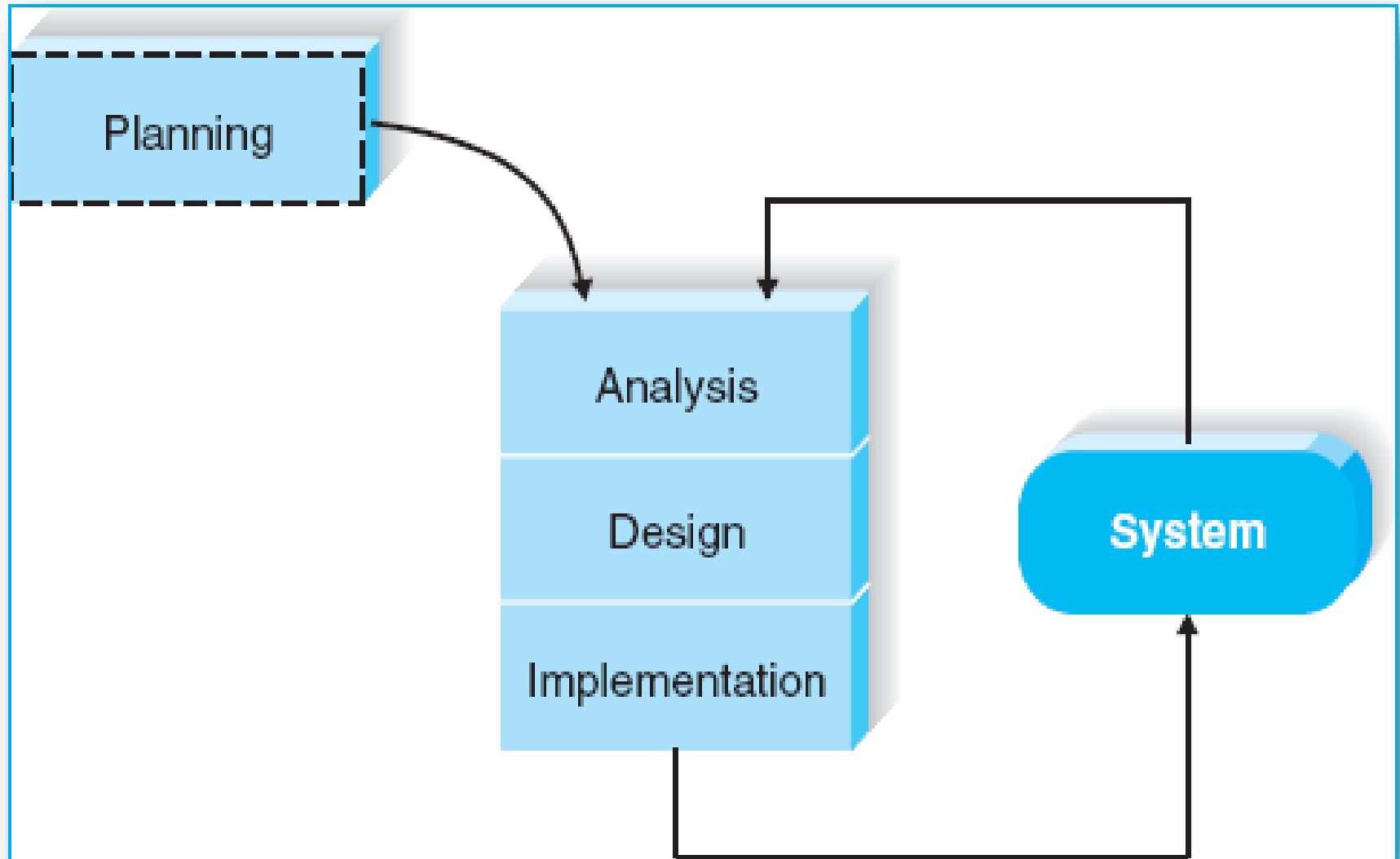
“Core Values” of XP

1. **Communication** – All to All
2. **Simplicity** – **KISS**, refactoring
3. **Feedback** – Embrace **Change**
4. **Courage** – **Quality** First, test and efficient coding

Extreme Programming (XP)

1. **User Stories** about system do
2. **Code small** program
3. **User Feedback**
4. **Repeat**
5. **Standards** are important
 - Naming conventions
 - Coding practices

Extreme Programming



Selecting the Appropriate Methodology

1. Clarity of User Requirements
2. Familiarity with Technology
3. System Complexity
4. System Reliability
5. Short Time Schedules
6. Schedule Visibility

Selecting the Right Methodology

Ability to Develop Systems	Structured Methodologies			RAD Methodologies		Agile Methodologies
	Waterfall	Parallel	Phased	Prototyping	Throwaway Prototyping	XP
with Unclear User Requirements	Poor	Poor	Good	Excellent	Excellent	Excellent
with Unfamiliar Technology	Poor	Poor	Good	Poor	Excellent	Poor
that are Complex	Good	Good	Good	Poor	Poor	Poor
that are Reliable	Good	Good	Good	Poor	Poor	Good
with a Short Time Schedule	Poor	Good	Excellent	Excellent	Good	Excellent
with Schedule Visibility	Poor	Poor	Excellent	Excellent	Good	Good

Selecting Methodology

Suppose you are an analyst for the Roanoke Software Consulting Company (RSCC), a large consulting firm with offices around the world. **The company wants to build a new knowledge management system that can identify and track the expertise of individual consultants anywhere in the world based on their education and the various consulting projects on which they have worked.** Assume that this is a new idea that never done before been attempted in RSCC or elsewhere. RSCC has an international network, but the offices in each country may use somewhat different hardware and software. RSCC management wants the system up and running within a year.

5 Problems in the Software Development

1. **Poor Requirements** - if requirements are unclear, incomplete, too general, and not testable, there may be problems.
2. **Unrealistic Schedule** - if too much work is crammed in too little time, problems are inevitable.
3. **Inadequate Testing** - no one will know whether or not the software is any good until customers complain or systems crash.
4. **Featuritis** - requests to add on new features after development goals are agreed on.
5. **Miscommunication** - if developers don't know what's needed or customer's have erroneous expectations, problems can be expected.

5 Solutions in the Software Development

1. **Solid Requirements** - clear, complete, detailed, cohesive, attainable, testable requirements that are agreed to by all players. In 'agile'-type environments, continuous close coordination with customers/end-users is necessary to ensure that changing/emerging requirements are understood.

5 Solutions in the Software Development

2. **Realistic Schedules** - allow adequate time for planning, design, testing, bug fixing, re-testing, changes, and documentation; personnel should be able to complete the project without burning out.

5 Solutions in the Software Development

3. **Adequate Testing** - start testing early on, re-test after fixes or changes, plan for adequate time for testing and bug-fixing. 'Early' testing could include static code analysis/testing, test-first development, unit testing by developers, built-in testing and diagnostic capabilities, automated post-build testing, etc.

5 Solutions in the Software Development

4. **Stick to Initial Requirements where Feasible** - be prepared to defend against excessive changes and additions once development has begun, and be prepared to explain consequences. If changes are necessary, they should be adequately reflected in related schedule changes. If possible, work closely with customers/end-users to manage expectations. In 'agile'-type environments, initial requirements may be expected to change significantly, requiring that true agile processes be in place and followed.

5 Solutions in the Software Development

5. **Communication** - require walkthroughs and inspections when appropriate; make extensive use of group communication tools - groupware, wiki's, bug-tracking tools and change management tools, intranet capabilities, etc.; ensure that information/documentation is available and up-to-date - preferably electronic, not paper; promote teamwork and cooperation; use prototypes and/or continuous communication with end-users if possible to clarify expectations.

The Unified Modelling Language (UML)

System Analysis Design Paradigm

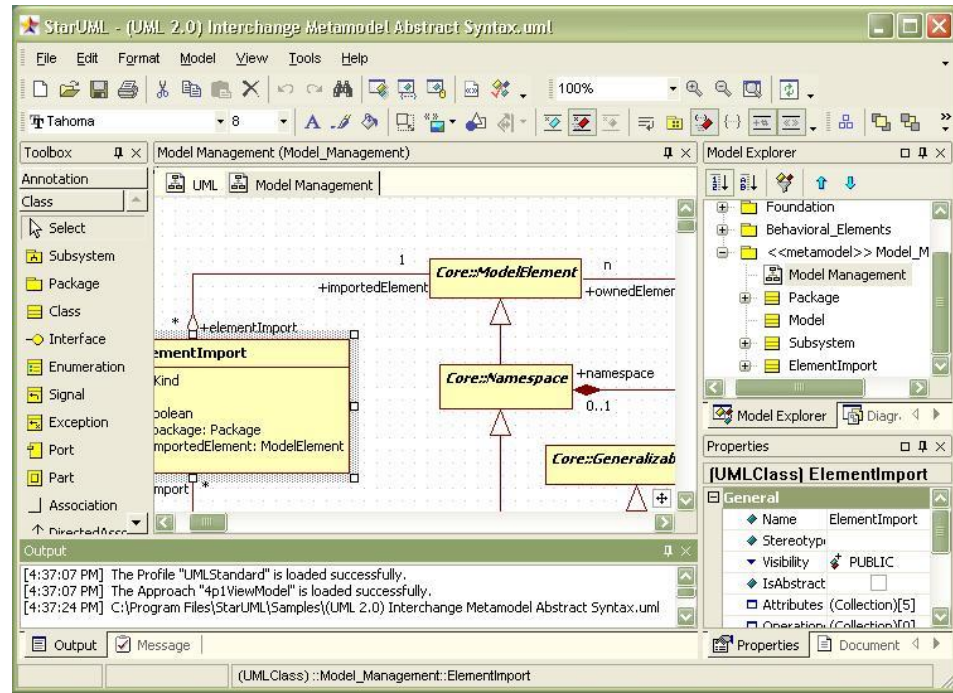
- **Data**-oriented: ERD
- **Process**-oriented: DFD+Flowchart
- **Object**-oriented (data + process): UML

Sejarah UML



Booch, Jacobson, Rumbaugh

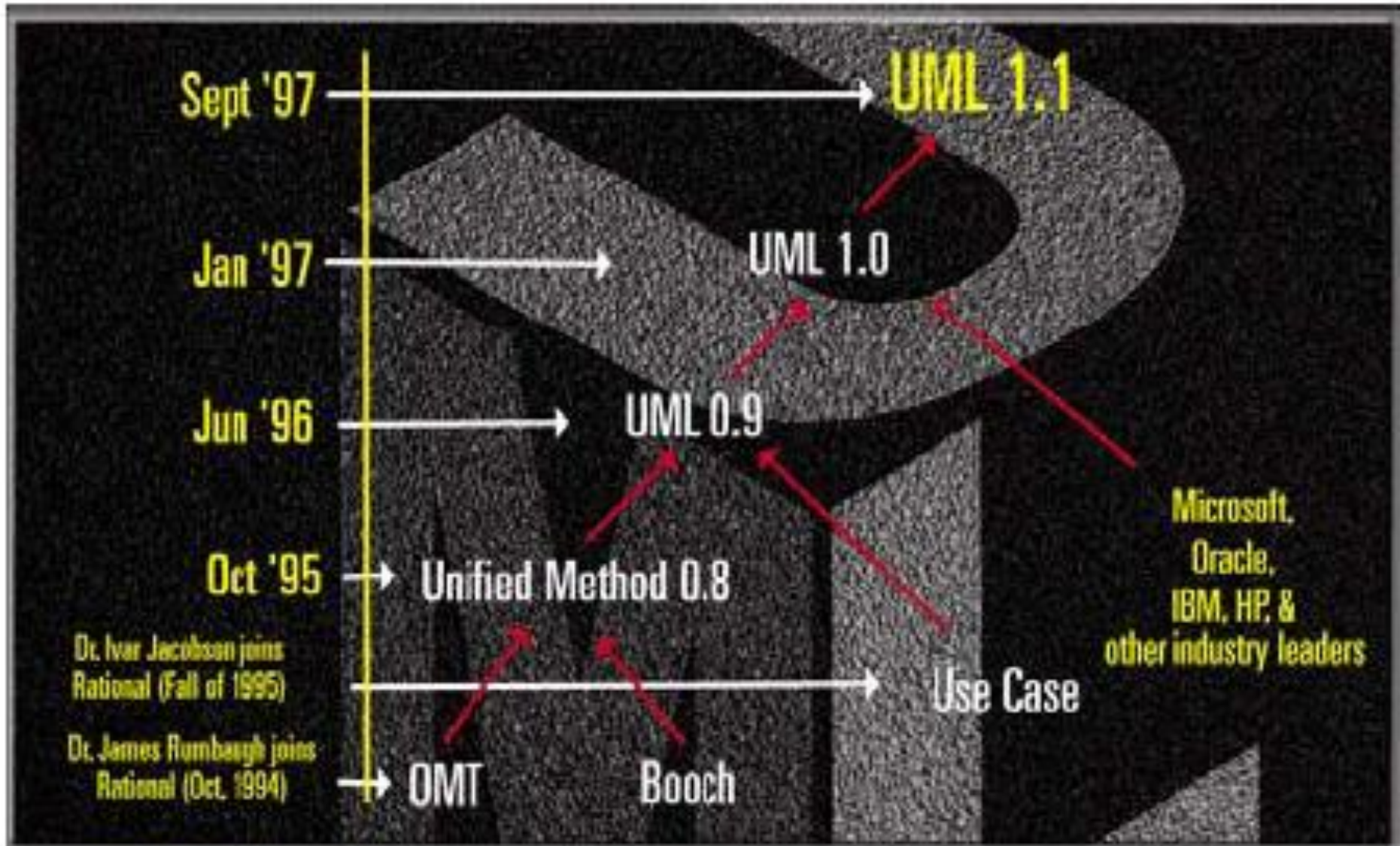
- In the 90s many people creating OO diagramming languages
- Three different ones created by Grady Booch, Ivar Jacobson, James Rumbaugh
- Joined forces with Rational (company) to create Unified Modeling Language (UML)





Sejarah UML

2003 → UML 2.0



What is the UML?

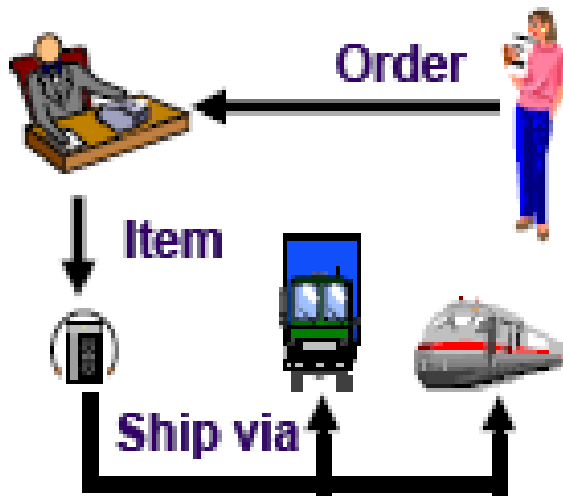
- UML: **Unified Modeling Language**
- UML can be used for **modeling all processes in the development life cycle** and across different implementation technologies (technology and language independent)
- UML is the **standard language** for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system

What is the UML?

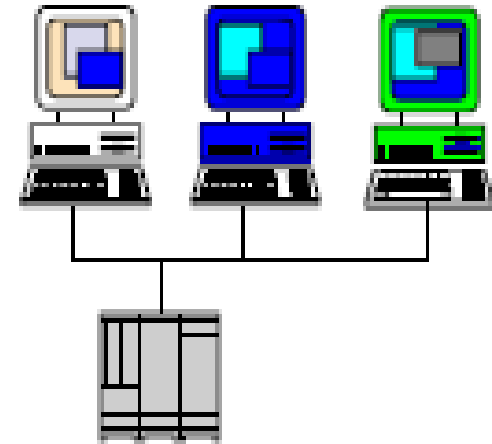
- UML is an **object-oriented modeling language**
 - semi-formal (UML 2.0 added much more formality)
 - **process independent**
 - ✓ UML can be used with a variety software development process models
 - customizable and extensible
 - model structure (static) and behavioral (dynamic) aspects of a system
- UML is a **communication tool** – for the team, and other stakeholders
 - A standard way to create a visual model

Why Modeling?

Modeling captures **essential parts of the system**
(James Rumbaugh)



Business Process

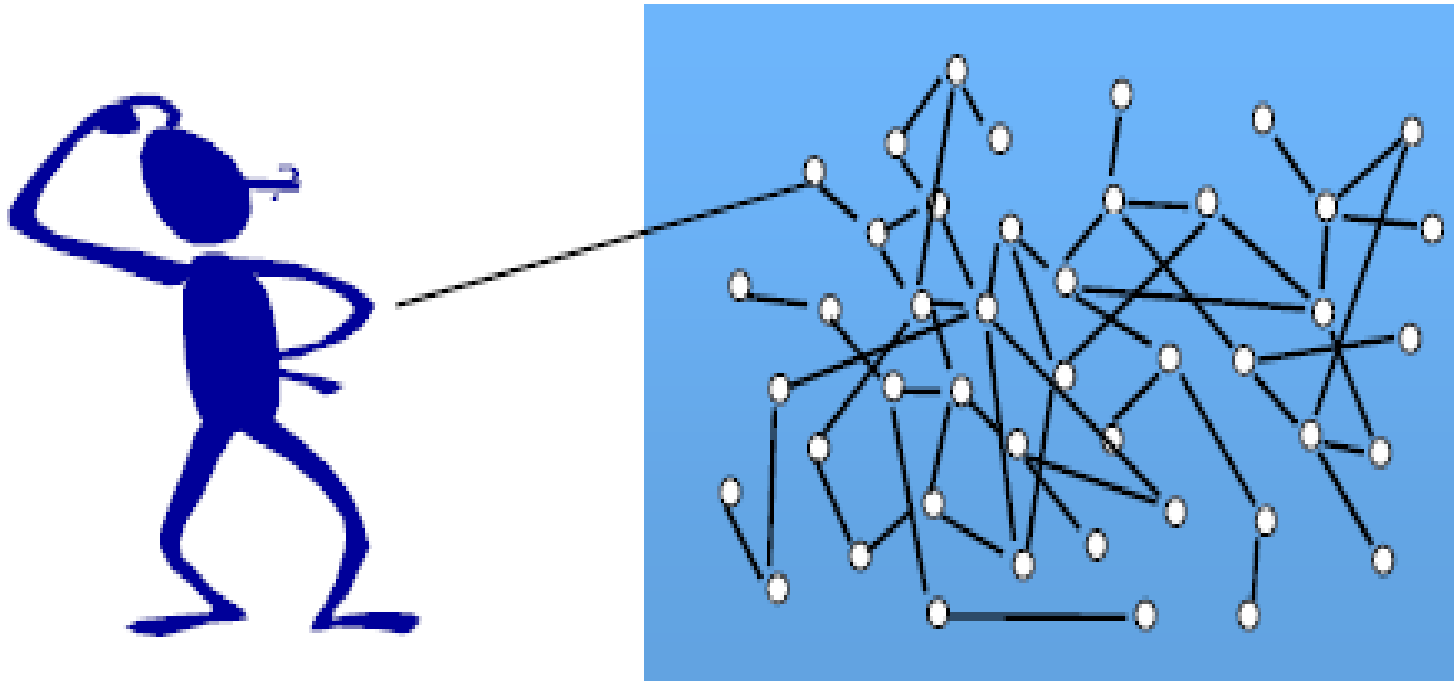


Computer System

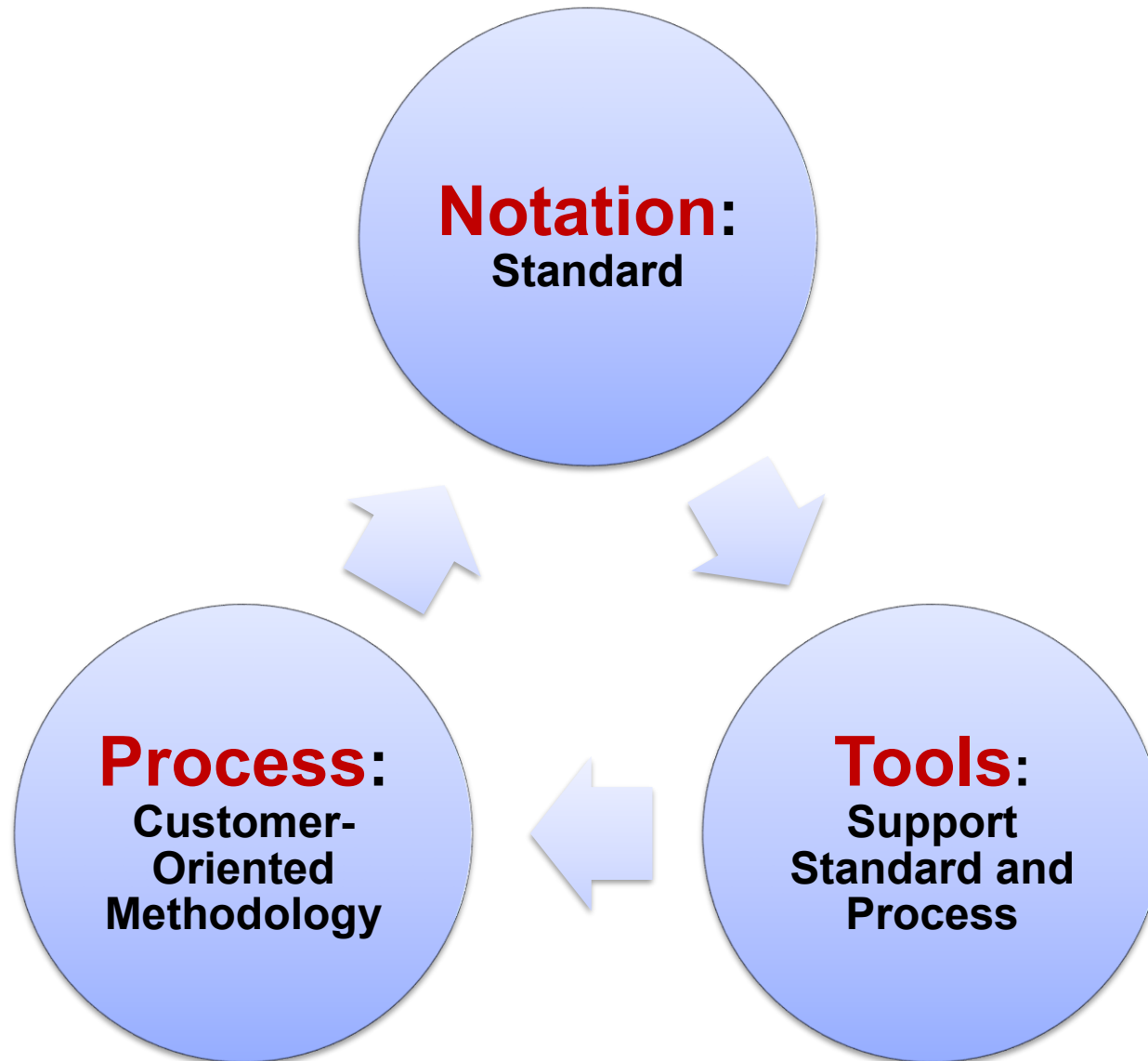
Visual Modeling is modeling using
standard graphical notations

Modeling Captures Business Process

Use Case Analysis is a technique to
capture business process from user's perspective



The Triangle of Success in Software Dev.



UML Tools

- Rational Rose
- Visual Paradigm
- Enterprise Architect
- ArgoUML
- Star UML
- Power Designer
- Netbeans UML Plugin
- Tools mampu mengubah class diagram menjadi source code dan sebaliknya

UML 2.0

UML version 2.0 has 14 diagrams in 2 major groups:

1. **Structure** Diagrams : diagram yang menyimpan struktur OO dan bersifat statis
2. **Behavior** Diagrams : diagram yang menggambarkan perilaku dinamis dari OO yang akan mengakses properti (atribut dan metode) dari diagram struktur

UML 2.0 Diagram

Diagram Name	Used to	Primary Phase
Structure Diagrams		
Class	Illustrate the relationships between classes modeled in the system.	Analysis, Design
Object	Illustrate the relationships between objects modeled in the system. Used when actual instances of the classes will better communicate the model.	Analysis, Design
Package	Group other UML elements together to form higher level constructs.	Analysis, Design, Implementation
Deployment	Show the physical architecture of the system. Can also be used to show software components being deployed onto the physical architecture.	Physical Design, Implementation
Component	illustrate the physical relationships among the software components.	Physical Design, Implementation
Composite Structure	Illustrate the internal structure of a class, i.e., the relationships among the parts of a class.	Analysis, Design
Behavioral Diagrams		
Activity	Illustrate business workflows independent of classes, the flow of activities in a use case, or detailed design of a method.	Analysis, Design
Sequence	Model the behavior of objects within a use case. Focuses on the time-based ordering of an activity.	Analysis, Design
Communication	Model the behavior of objects within a use case. Focuses on the communication among a set of collaborating objects of an activity.	Analysis, Design
Interaction Overview	Illustrate an overview of the flow of control of a process.	Analysis, Design
Timing	Illustrate the interaction that takes place among a set of objects and the state changes in which they go through along a time axis.	Analysis, Design
Behavioral State Machine	Examine the behavior of one class.	Analysis, Design
Protocol State Machine	Illustrates the dependencies among the different interfaces of a class.	Analysis, Design
Use-Case	Capture business requirements for the system and to illustrate the interaction between the system and its environment.	Analysis

UML Structure Diagrams

Represent the data and static relationships in an information system

1. **Class** Diagram
2. **Object** Diagram
3. **Package** Diagram
4. **Deployment** Diagram
5. **Component** Diagram
6. **Composite** Structure Diagram

Structure Diagrams

1. **Class** Diagrams: diagram hubungan antar kelas
 - Common vocabulary used by analyst and users
 - Represent things (employee, paycheck,...)
 - Shows the relationships between classes
2. **Object** Diagrams: diagram hubungan antar obyek
 - Similar to class diagrams
 - Instantiation of a class diagram
 - Relationships between objects
3. **Package** Diagrams : paket/direktori class/benda
 - Group UML elements together to form higher level constructs

Structure Diagrams

4. Deployment Diagrams

- Shows the physical architecture and software components of system
- For example, network nodes

5. Component Diagrams

- Physical relationships among software components
- Example – Client/Server (Which machines run which software)

6. Composite Structure

- Illustrates internal structure of a complex class

UML Behavior Diagrams

Depict the dynamic relationships among the instances or objects that represent the business information system

1. **Activity** Diagram
2. **Sequence** Diagram
3. **Communication** Diagram
4. **Interaction** Overview Diagram
5. **Timing** Diagram
6. **Behavior State** Machine
7. **Protocol State** Machine
8. **Use Case** Diagrams

Behavior Diagrams

1. Activity Diagrams

- Model processes in an information system
- Example: Business workflows, business logic

2. Interaction Diagrams

- Shows interaction among objects

3. Sequence Diagrams

- Time-based ordering of the interaction

4. Communication Diagrams

- Communication among a set of collaborating objects of an activity

Behavior Diagrams

4. Interaction Overview Diagrams

- Overview of flow of control of a process

5. Timing Diagrams

- Show how an object changes over time

6. State Machines

- Examines behavior of one class
- Models the different states and state transitions an object can experience

7. Use-Case Diagrams

- Shows interaction between the system and environment
- Captures business requirements

UML Problems

- UML is modeling notation, it is **not a development process** or a methodology
 - UML driven development process?
- UML is **too complex, difficult to understand quickly**
 - Should we use all UML diagrams?

UML Driven Process

1. Model the organization **business process** with **activity diagrams** or **business process modeling notation (BPMN)**
2. Display the **boundary of a system** & its major functions using **use case diagram**
3. Illustrate **use case realizations** with **interaction diagrams** (**sequence diagram** and **activity diagrams**)
4. Represent a **static structure of a system** using **class and component diagram**
5. Reveal the **physical implementation architecture** with **deployment diagrams**

UML Process (Simple Version)

1. Business Process Modeling

1. Activity Diagram atau dengan Business Process Modelling (kalau kompleks)
2. Use Case Diagram

2. Interaction Diagram:

1. Sequence Diagram (Gunakan pola Boundary-Control-Entity)
2. Activity Diagram (menjelaskan dengan detail sequence diagram)

3. User Interface Model

- Buat UI design dari class Boundary Class

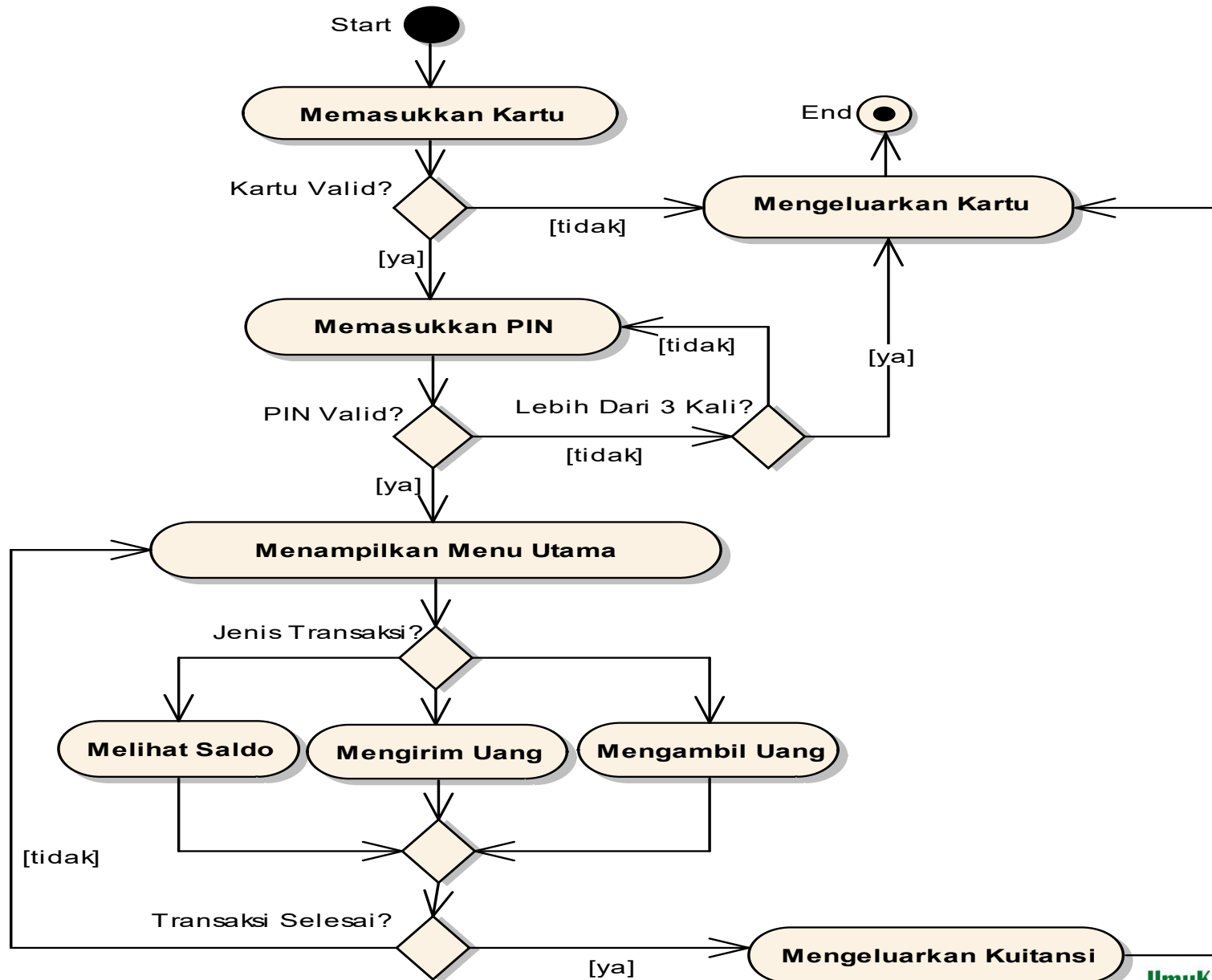
4. Entity-Relationship Model

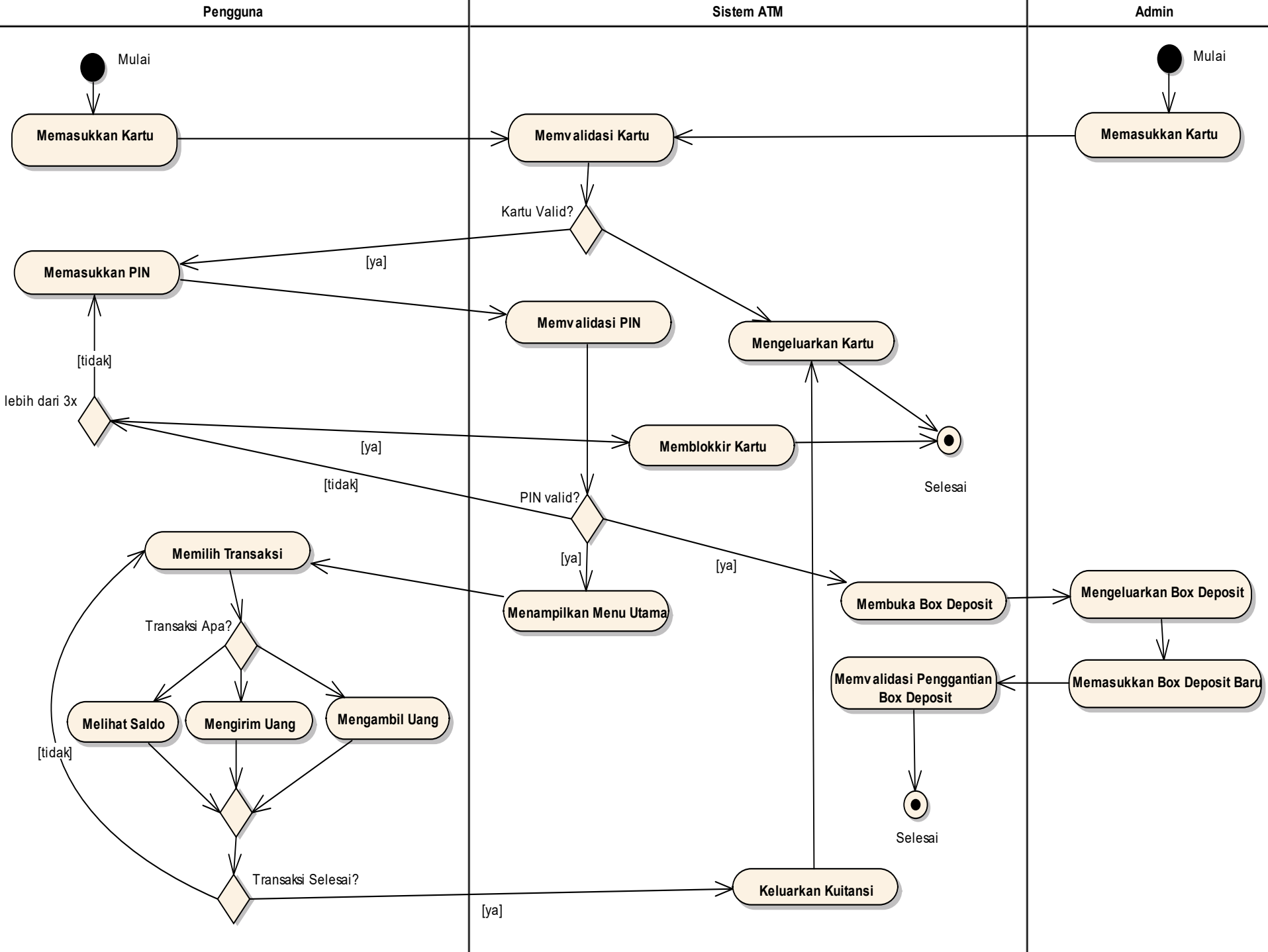
- Buat ER diagram dari Entity Class

5. Class Diagram

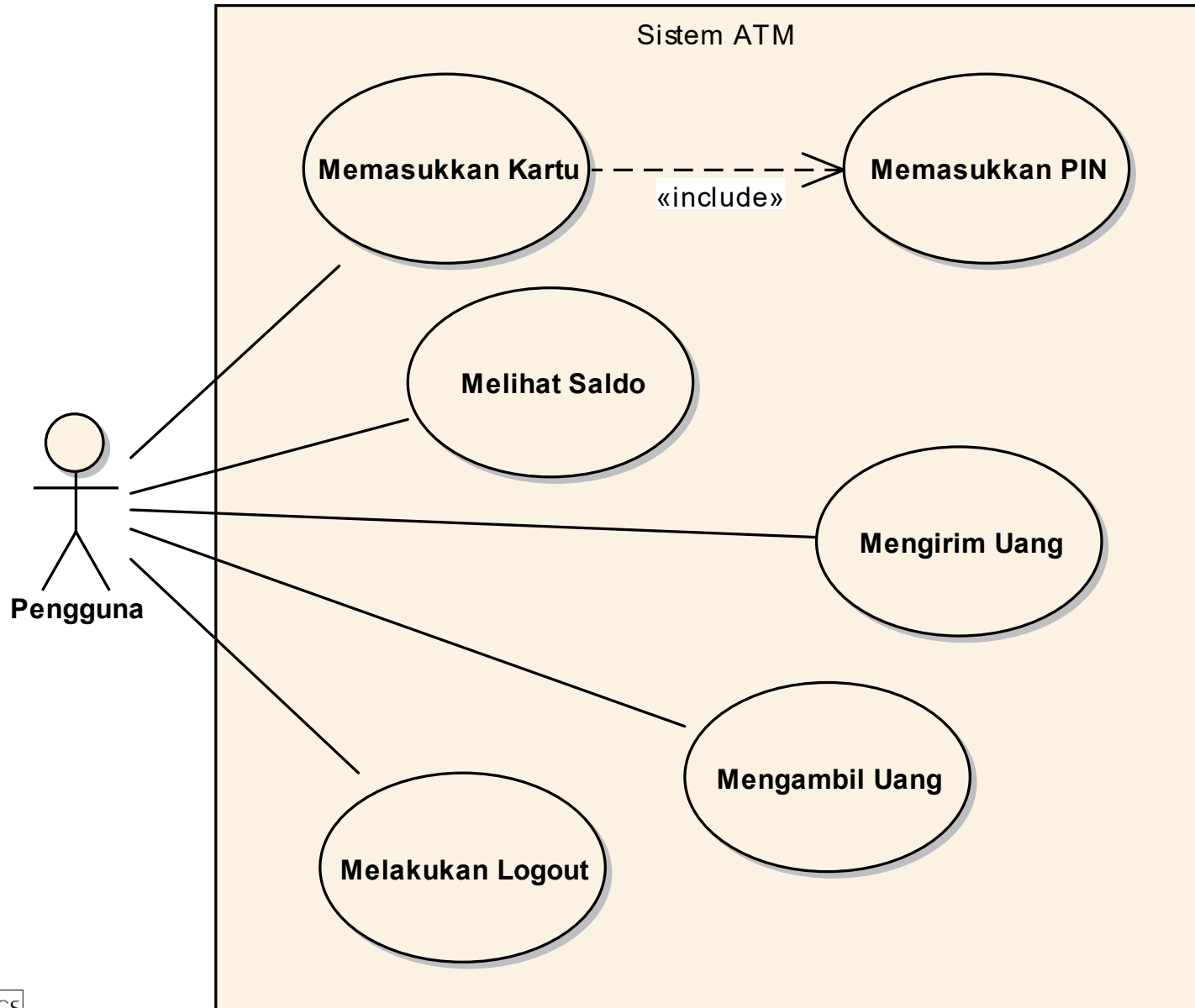
Susun story dari program kita pada class diagram

Activity Diagram (Business Process)

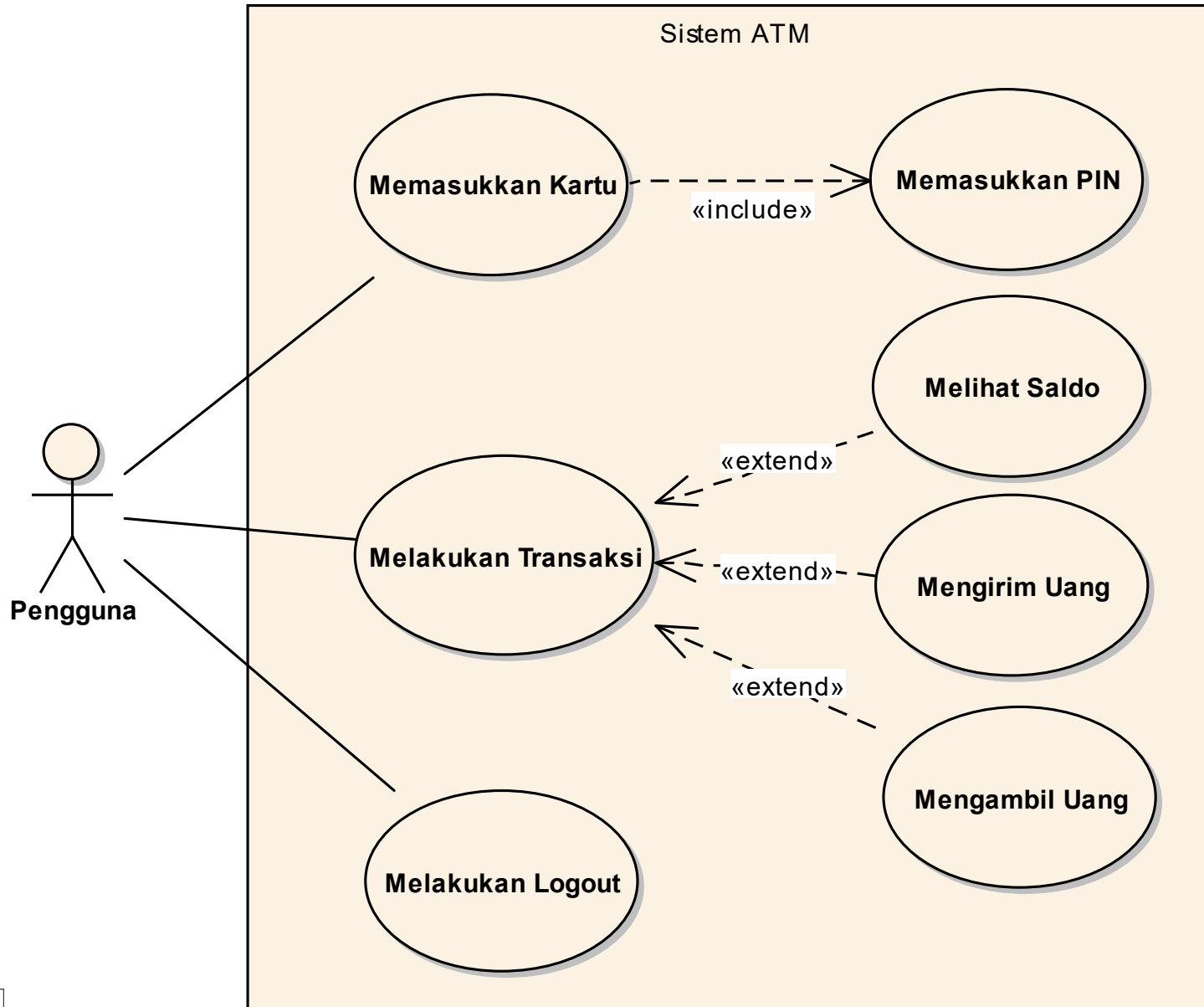




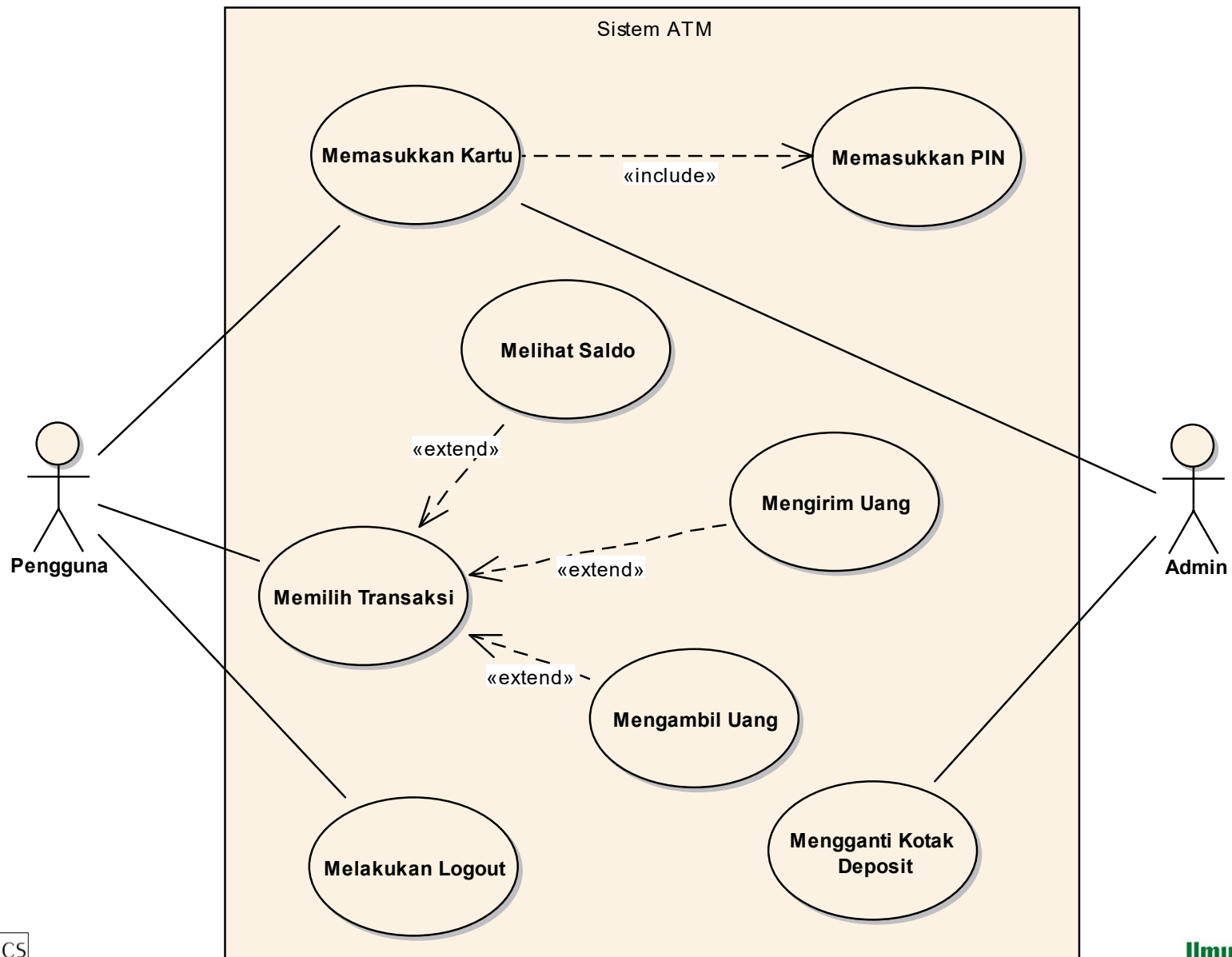
Use Case Diagram



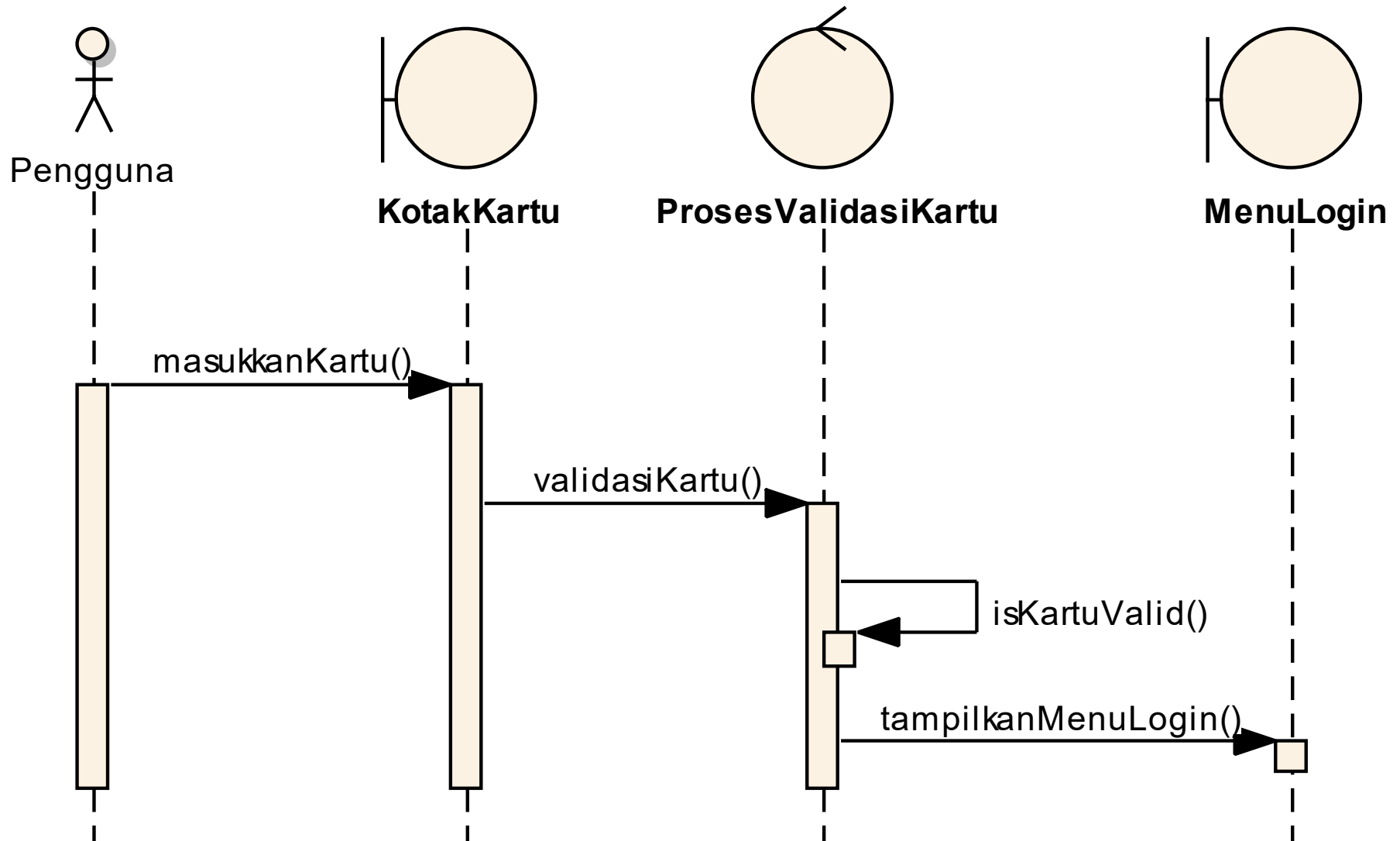
Use Case Diagram



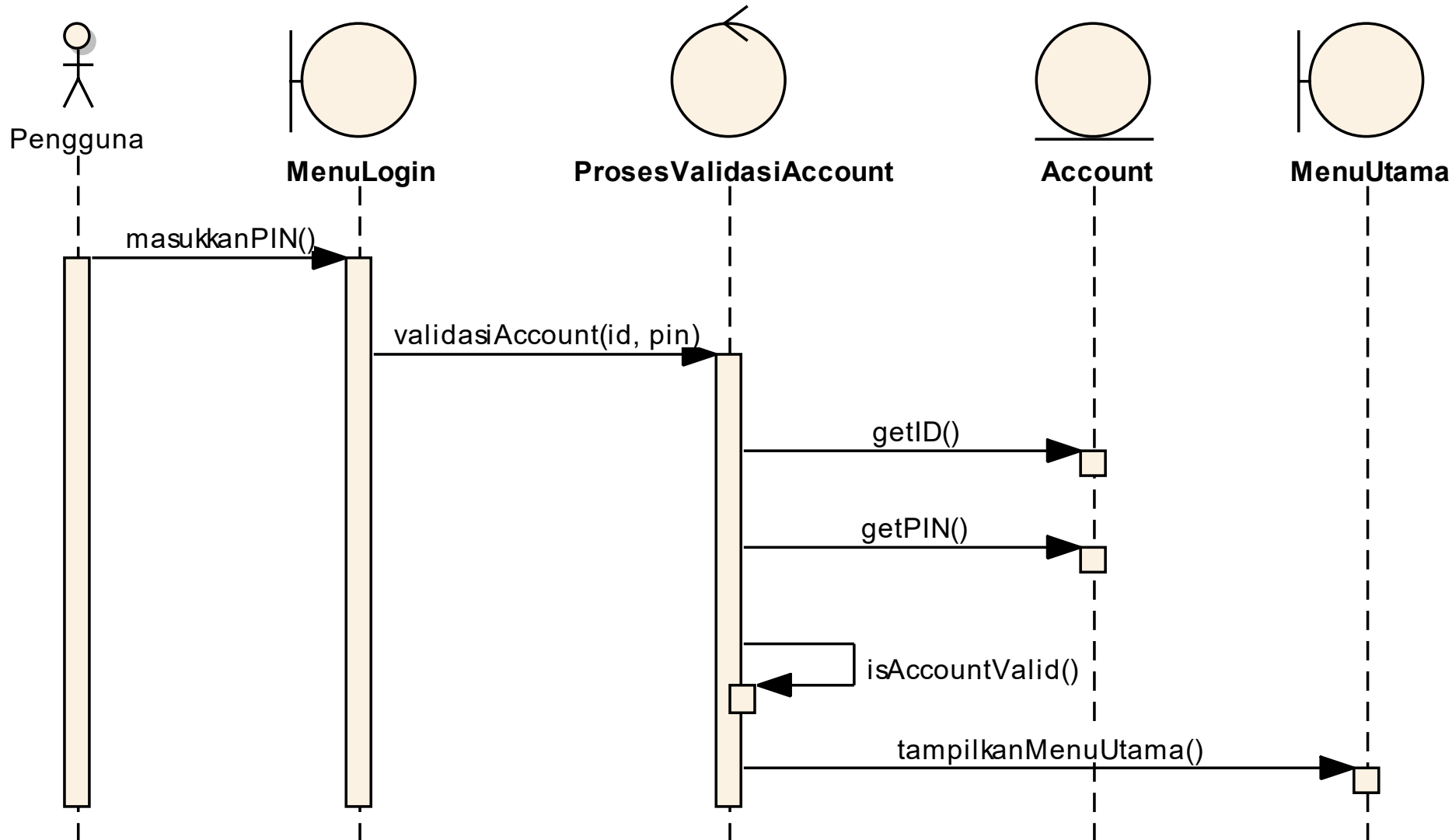
Use Case Diagram (Alternatif)



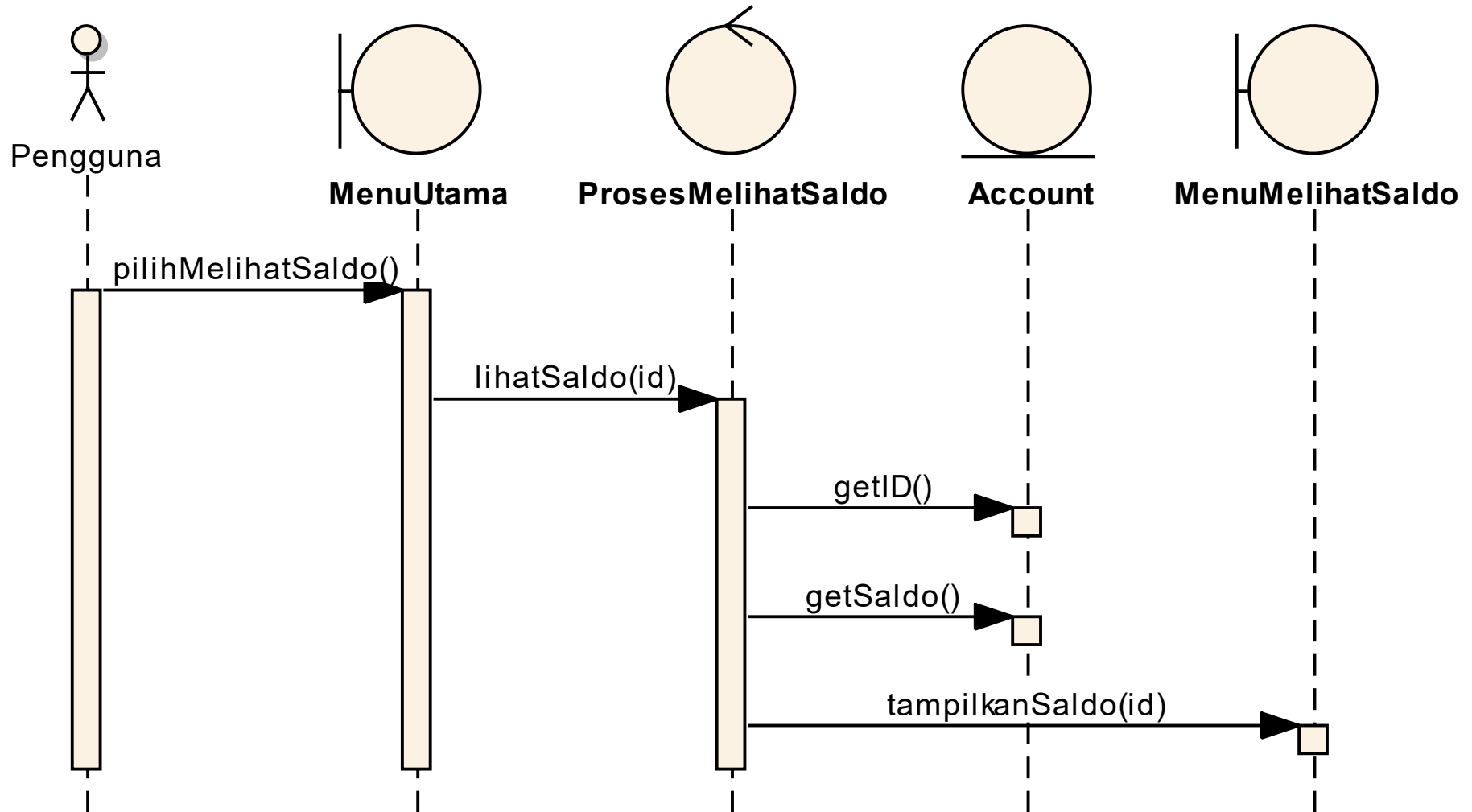
Sequence Diagram: Memasukkan Kartu



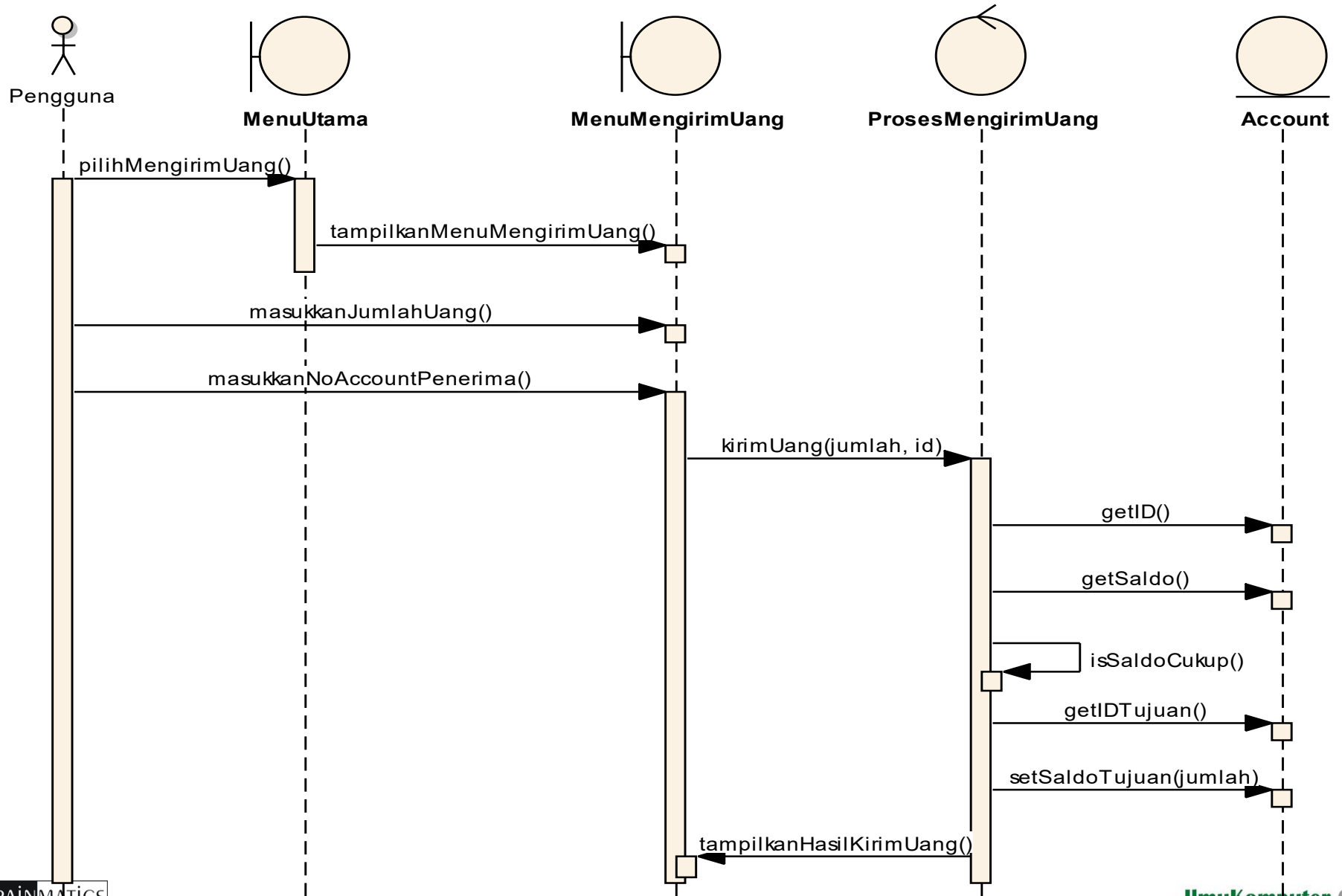
Sequence Diagram: Memasukkan PIN (konsep B,C,E)



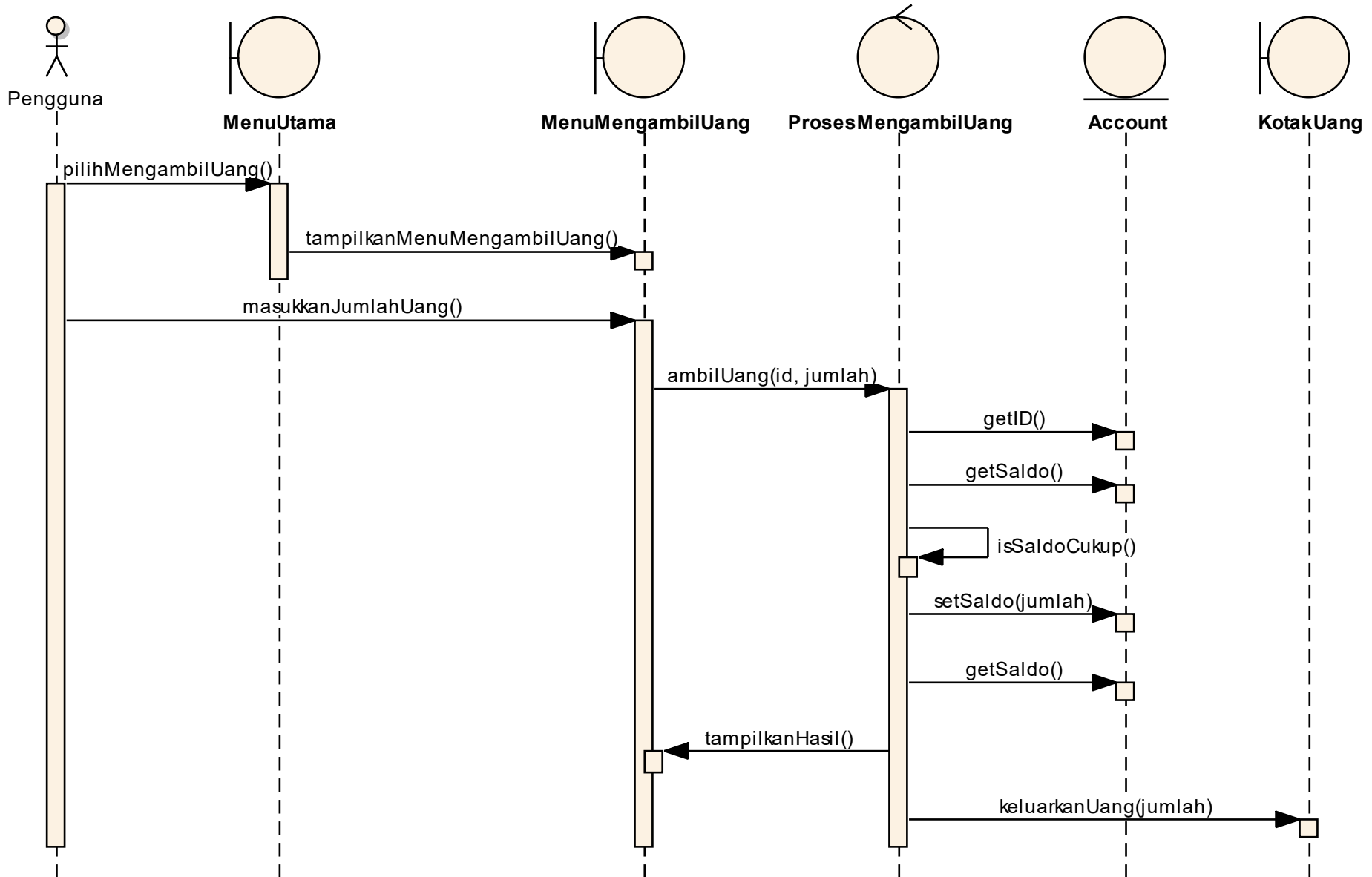
Sequence Diagram: Melihat Saldo



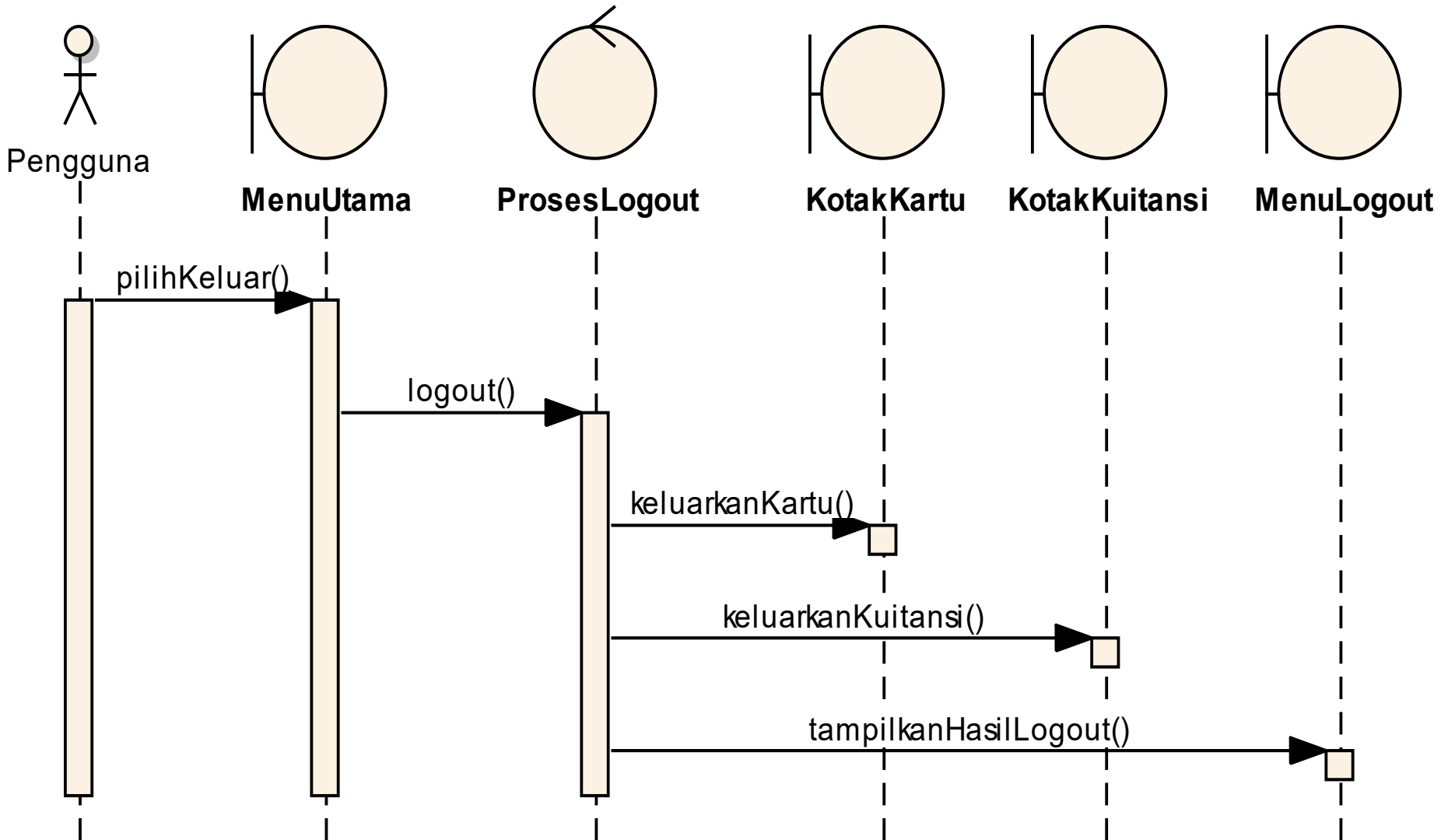
Sequence Diagram: Mengirim Uang



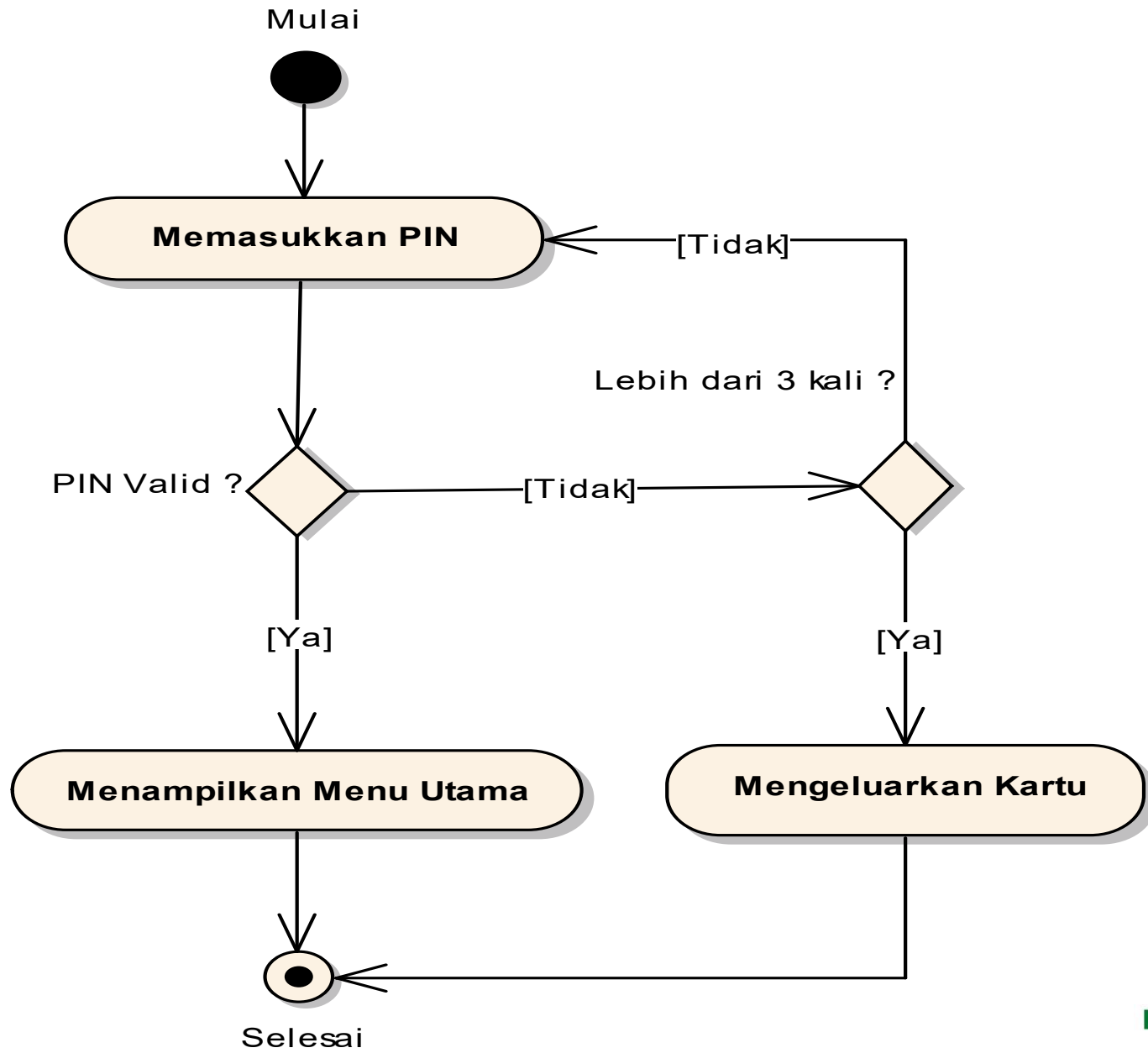
Sequence Diagram: Mengambil Uang



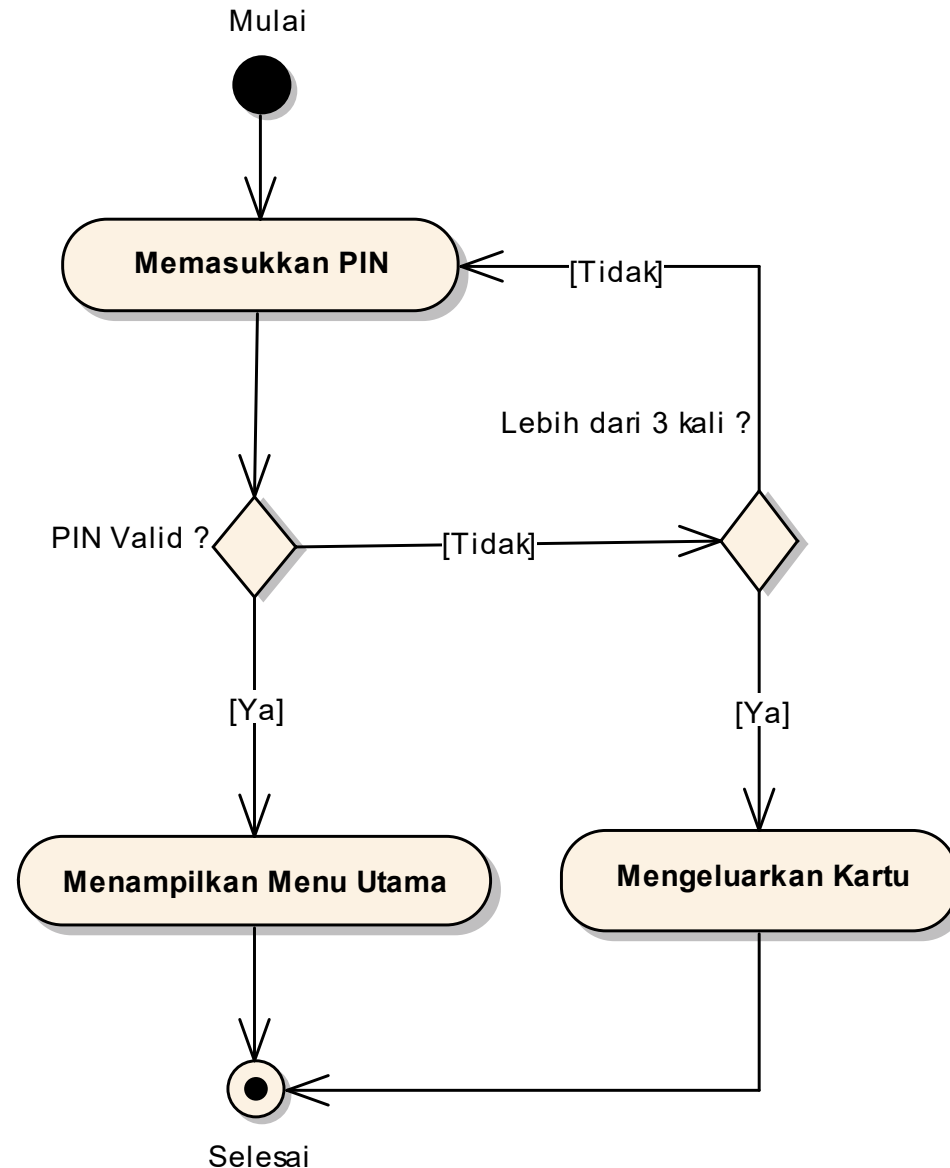
Sequence Diagram: Melakukan Logout



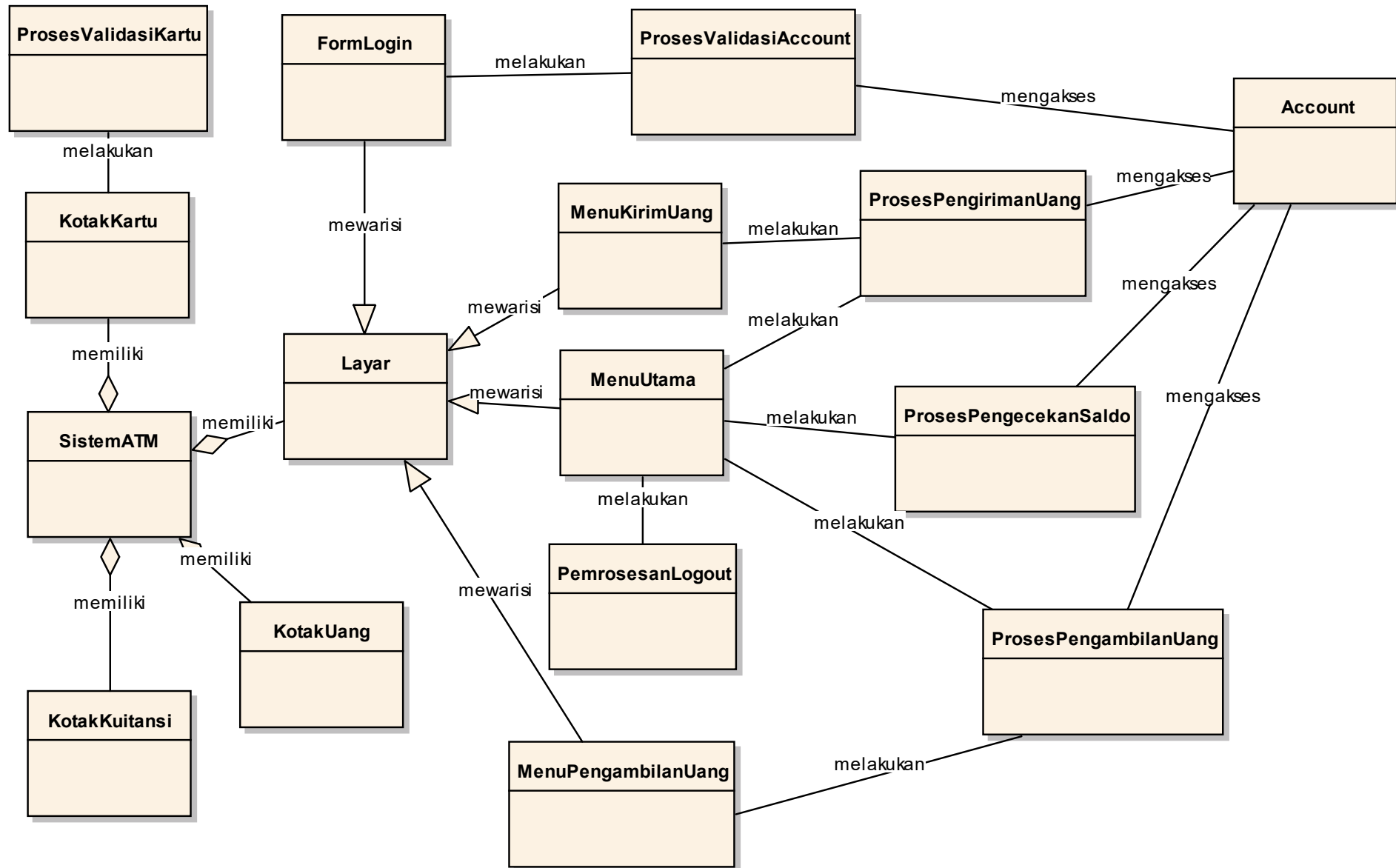
Activity Diagram: Memasukkan Kartu



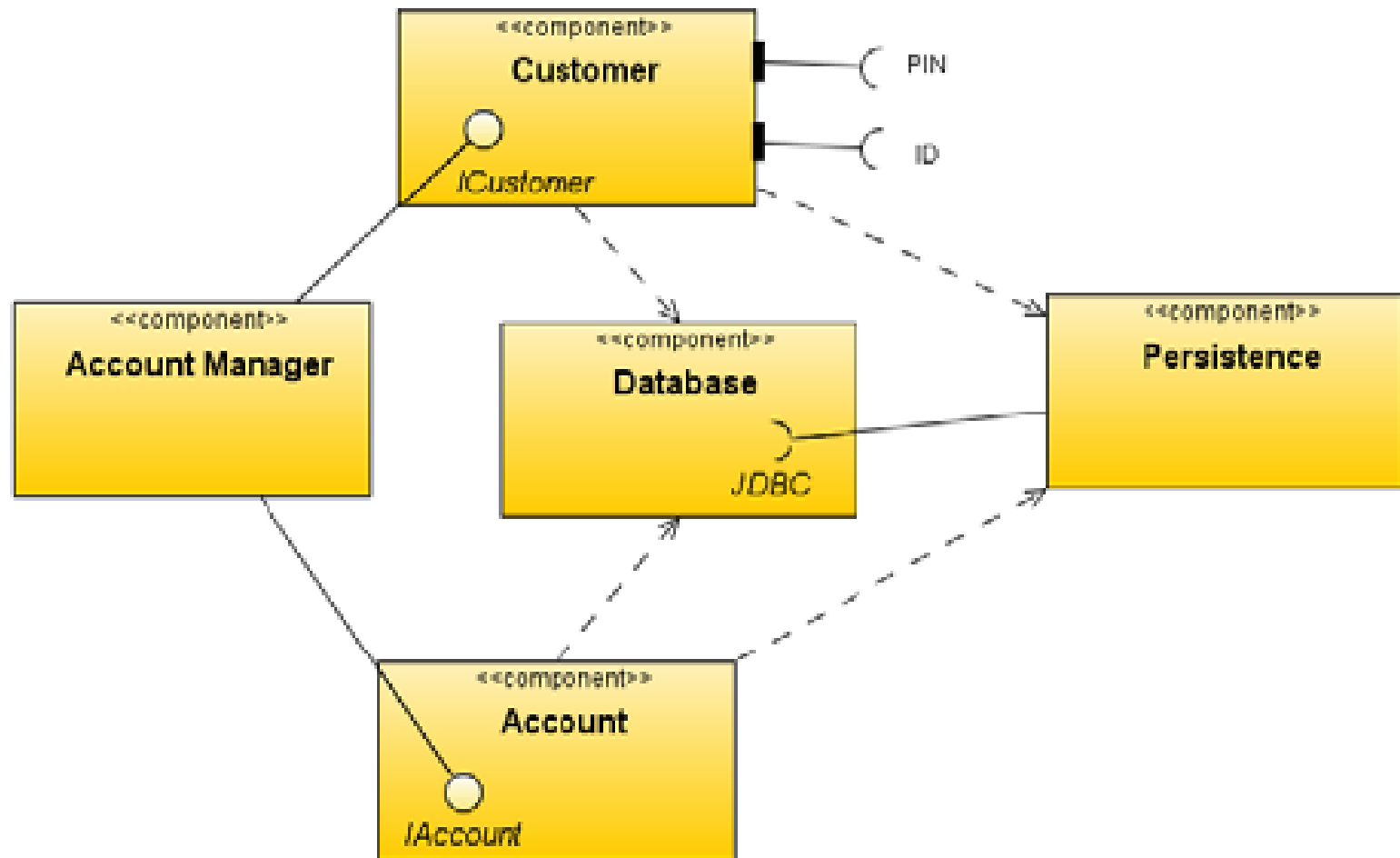
Activity Diagram: Memasukkan PIN



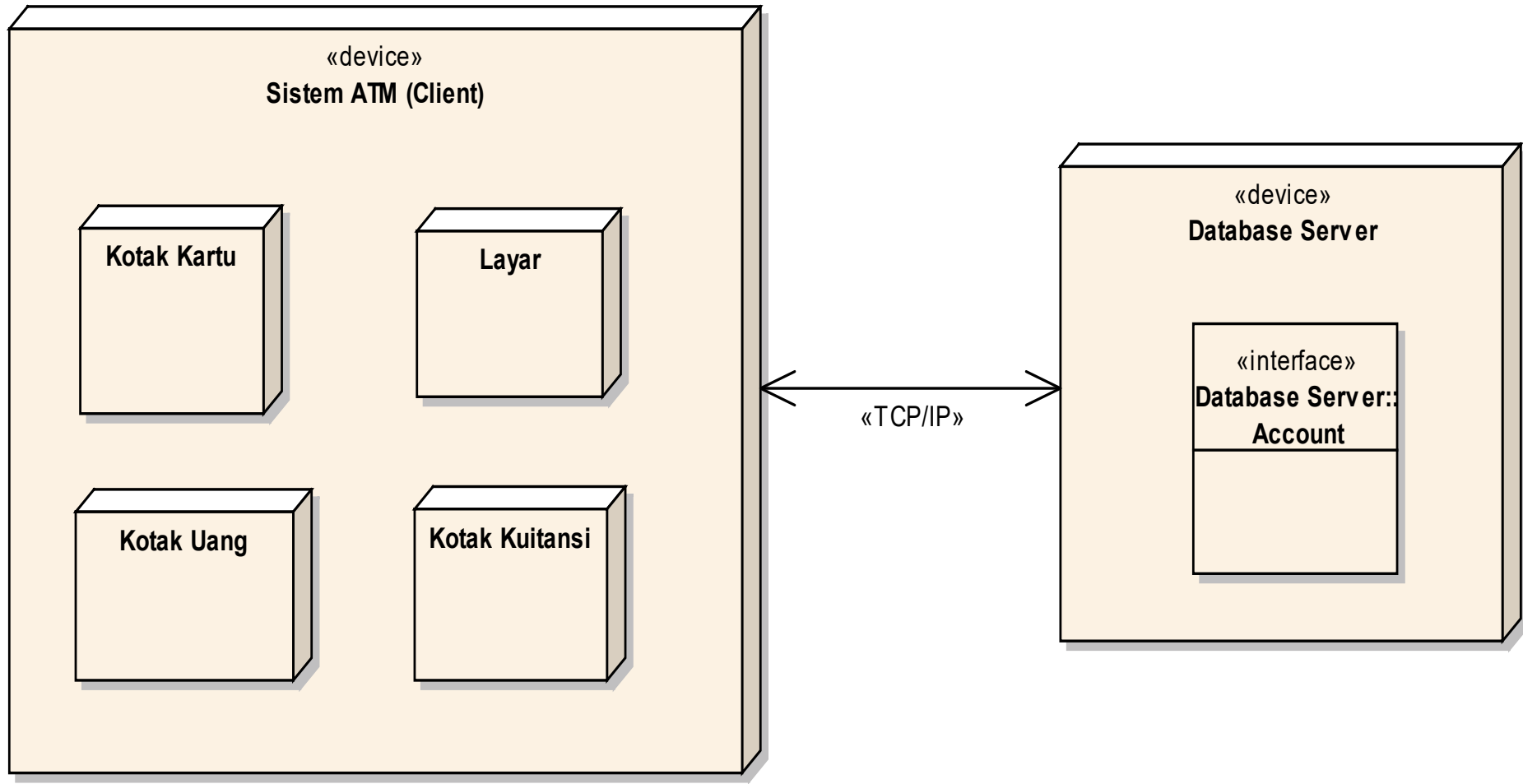
Class Diagram



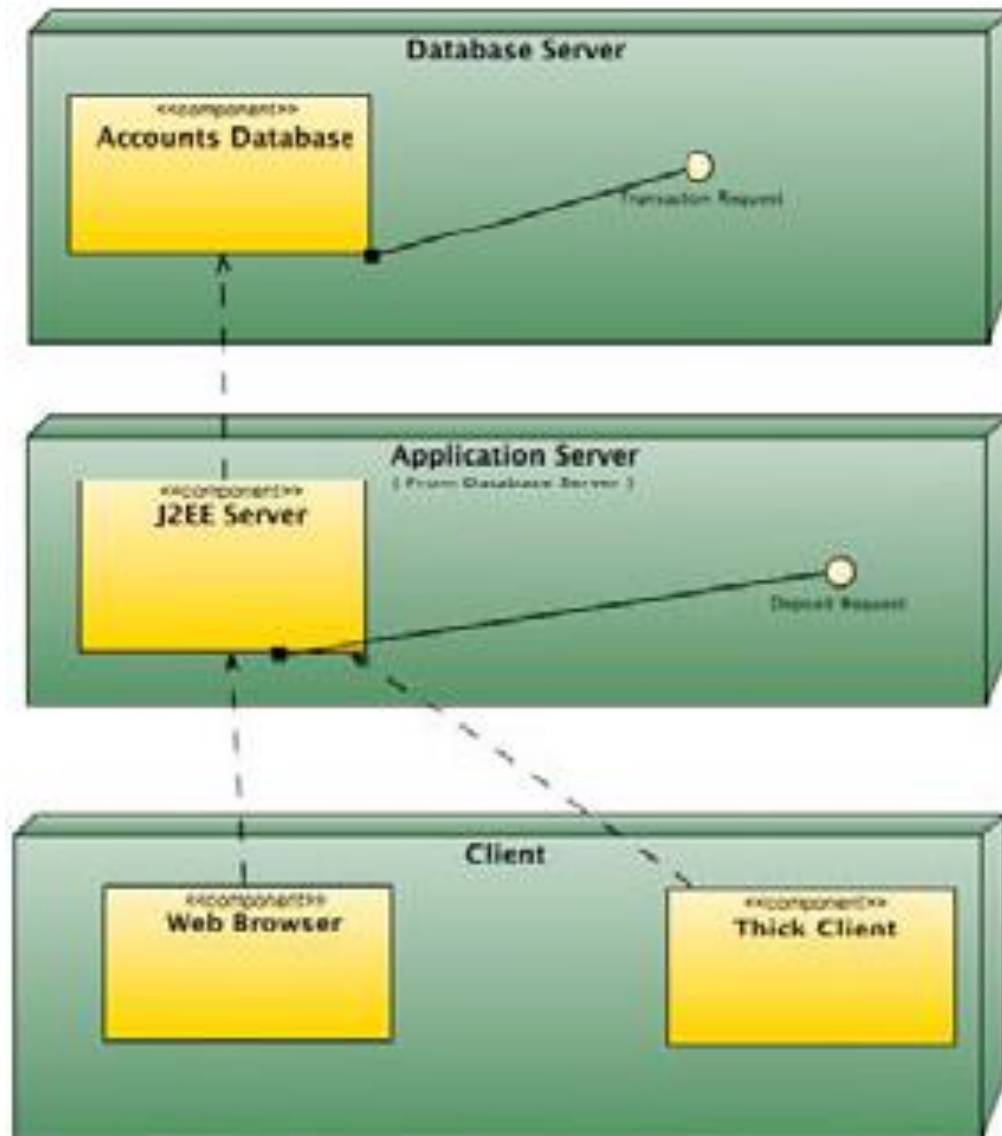
Component Diagram



Deployment Diagram (2 Tier)



Deployment Diagram (3 Tier)



Object-Oriented Paradigm

Pretest

1. Buat program java yang menampilkan di layar tulisan Halo Salemba
2. Pelajari dan pahami slide 1 dari folder romi-oop-01-oopconcepts-1nop2011

Konsep Dasar Pemrograman Berorientasi Objek

Class , Object, Method, Attribute

Berorientasi Objek?



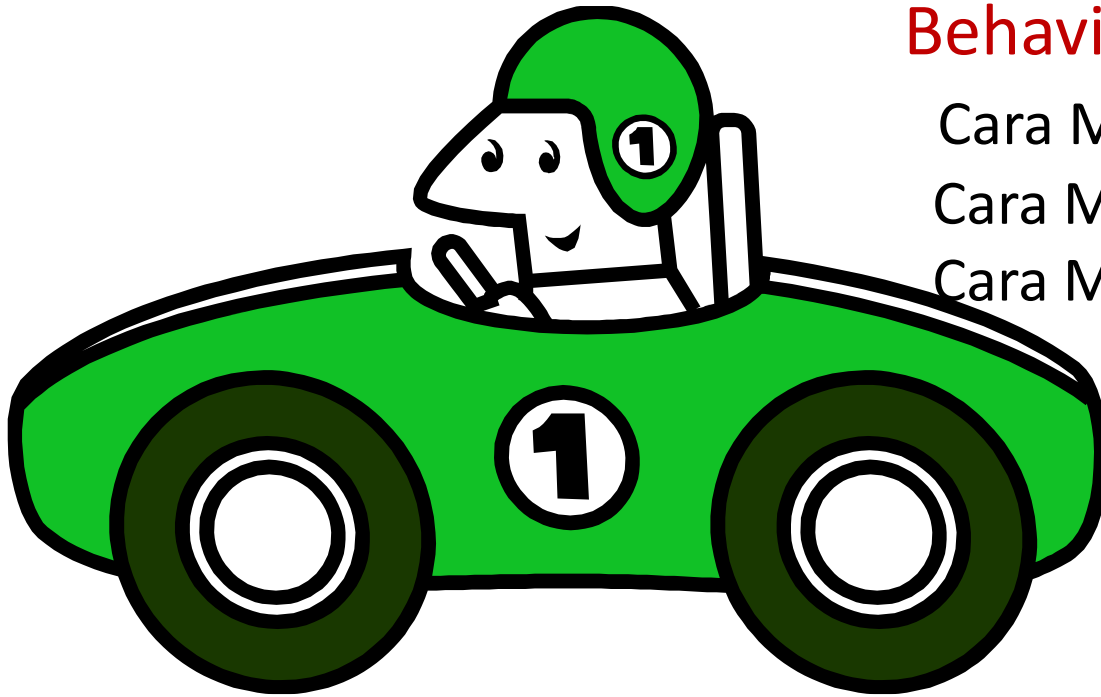
Attribute:

Topi, Baju, Jaket,
Tas Punggung,
Tangan, Kaki, Mata

Behavior:

Cara Jalan ke Depan
Cara Jalan Mundur
Cara Belok ke Kiri
Cara Memanjat

Berorientasi Objek?



Attribute (State):

Ban, Stir, Pedal Rem, Pedal Gas,
Warna, Tahun Produksi

Behavior:

Cara Menghidupkan Mesin
Cara Manjalankan Mobil
Cara Memundurkan Mobil

Attribute → Variable(Member)

Behavior → Method(Fungsi)

Latihan

1. Lakukan kegiatan berorientasi objek dengan target **Sepeda**
2. Pikirkan apa **atribut dan behavior** yang dimiliki oleh Sepeda tersebut



Perbedaan Class dan Object

- Class adalah **konsep** dan **deskripsi** dari sesuatu
- Object adalah **instance** dari Class

- Class: **mobil**
- Object: **mobilnya pak Joko, mobilku**

- Class seperti **cetakan kue**, dimana kue yg dihasilkan dari cetakan kue itu adalah **object**
- Warna kue bisa bermacam-macam meskipun berasal dari cetakan yang sama (**object memiliki sifat independen**)

Class = Method + Variable

Class Sepeda

gir

kecepatan

ubah kecepatan

ubah gir

variable

method

Object = Method + Variable **Bernilai**

Object Sepedaku

gir = 3

kecepatan = 10km/jam

ubah kecepatan (10)

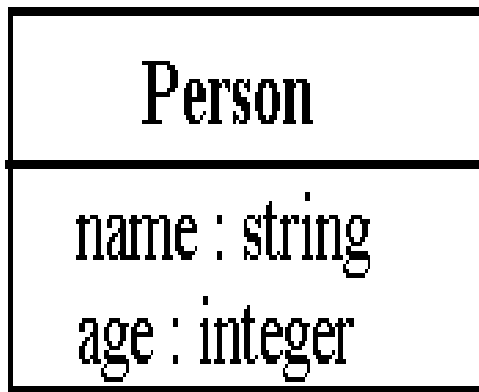
ubah gir (2)

instance
variable

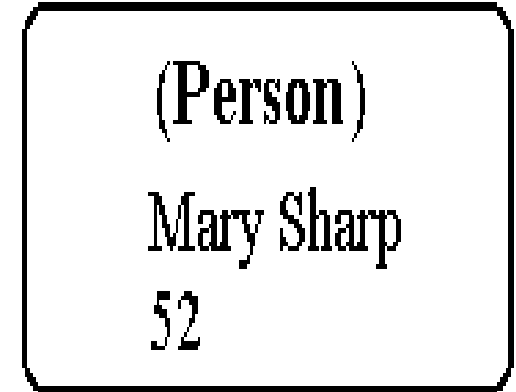
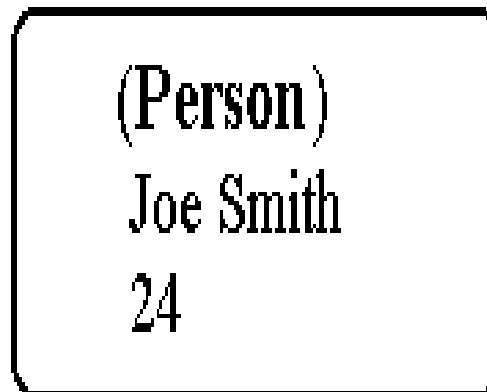
instance
method

Attribute

- **Variable** yang mengitari class, dengan **nilai datanya bisa ditentukan di object**
- Name, age, dan weight adalah atribut dari class Person



Class with Attributes



Objects with Values

Membuat Class, Object dan Memanggil Atribut

```
public class Mobil {  
    String warna;  
    int tahunProduksi;  
}
```

Mobil.java

```
public class MobilBeraksi{  
    public static void main(String[] args){  
        // Membuat object  
        Mobil mobilku = new Mobil();  
  
        /* memanggil atribut dan memberi nilai */  
        mobilku.warna = "Hitam";  
        mobilku.tahunProduksi = 2006;  
        System.out.println("Warna: " + mobilku.warna);  
        System.out.println("Tahun: " + mobilku.tahunProduksi);  
    }  
}
```

MobilBeraksi.java

Kelas vs main program

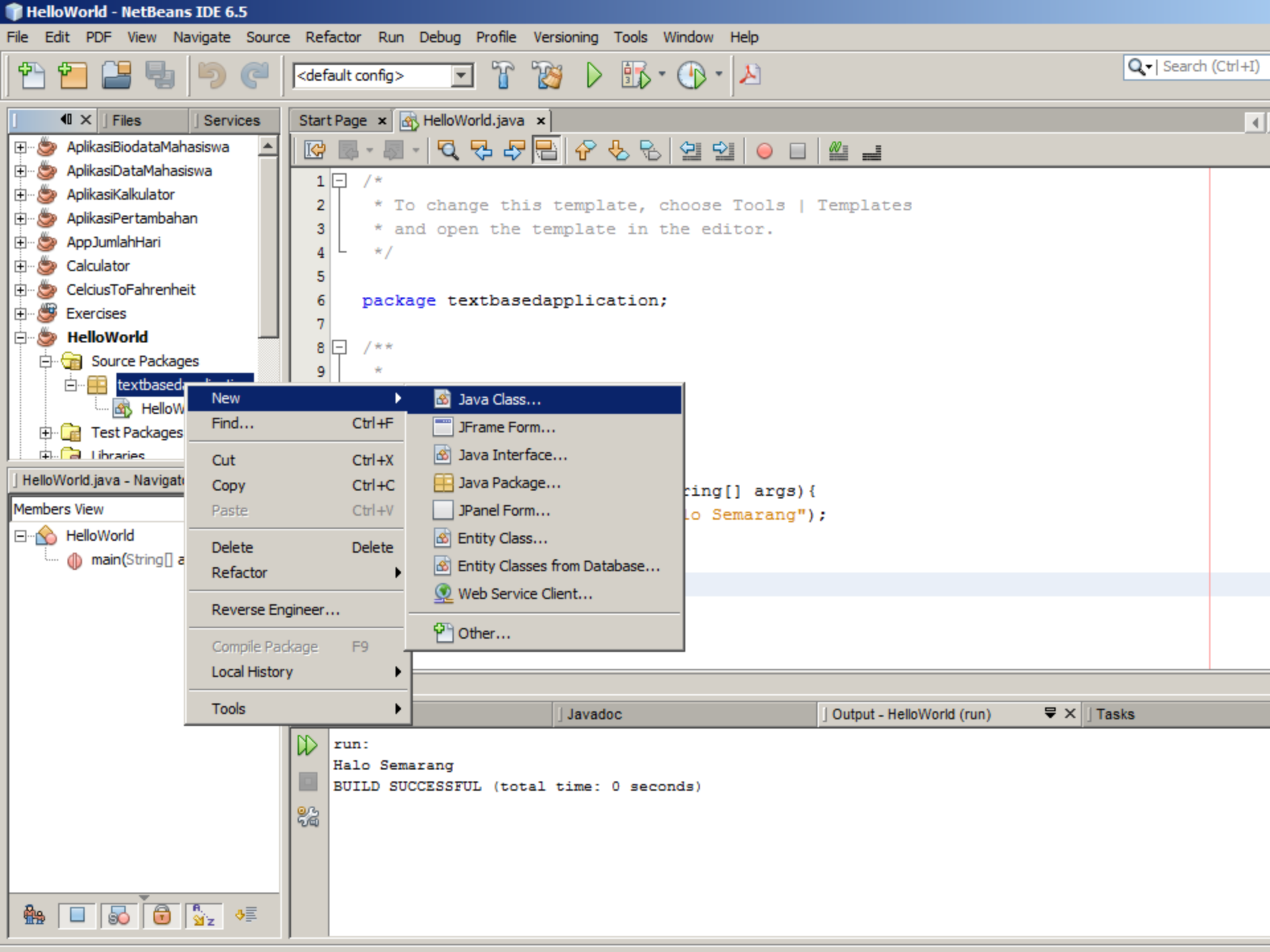
- Kelas : mendefinisikan atribut dan metode yang dimiliki oleh kelas
- Main program : melakukan :
 1. Instansiasi/penciptaan object dg kata kunci new
 2. object kemudian akan mengakses atribut baik untuk diinput value / state ke dalam atribut
 3. object bisa memanggil metode kelas yang relatif juga akan mengakses atribut.
- Melakukan closing dengan melakukan pencetakan

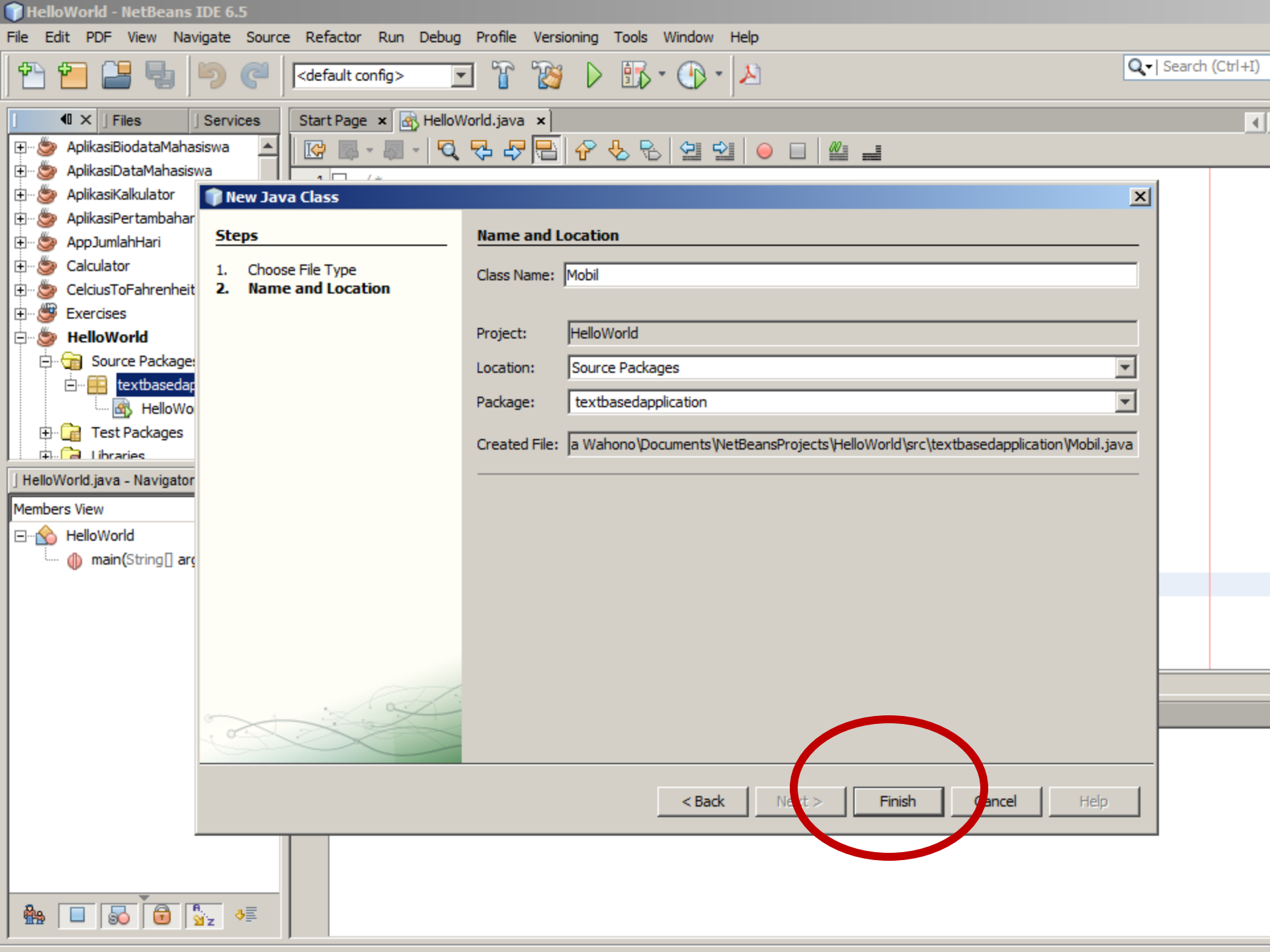
tugas

- Buat class (salah satu saja) : dosen, mahasiswa, mata kuliah, ruang kuliah, pembayaran, orangtua atau yang lain
- Lengkapi dengan atribut dan tipe atribut sesuai dengan persepsi anda
- Lengkapi dengan metode : input data(); metode cetakinfo(), UbahInfo() dan metode lain terserah persepsi anda
- Cetak dan print minggu depan kumpulan untuk kita coba maju lagi supaya ada kemajuan

Latihan: Membuat Program dgn Netbeans

1. Buka Netbeans IDE
2. Ikuti langkah berikut





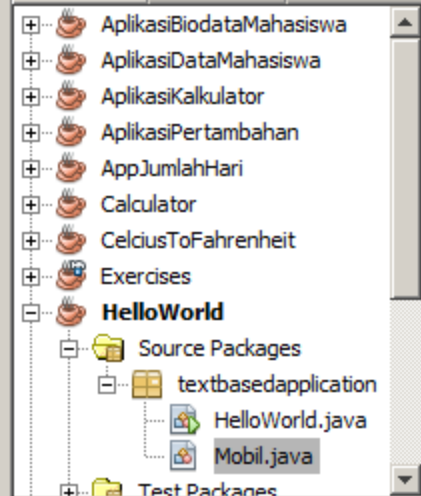


<default config>



Search (Ctrl+I)

Files Services



Navigator

Members View

Mobil

Start Page x HelloWorld.java x Mobil.java x



```
1  /*
2      * To change this template, choose Tools | Templates
3      * and open the template in the editor.
4      */
5
6  package textbasedapplication;
7
8  /**
9      *
10     * @author Romi Satria Mahono
11     */
12     public class Mobil {
13
14     }
15
```

1:1 INS

Usages

Javadoc

Output - HelloWorld (run)

Tasks

```
run:
Halo Semarang
BUILD SUCCESSFUL (total time: 0 seconds)
```



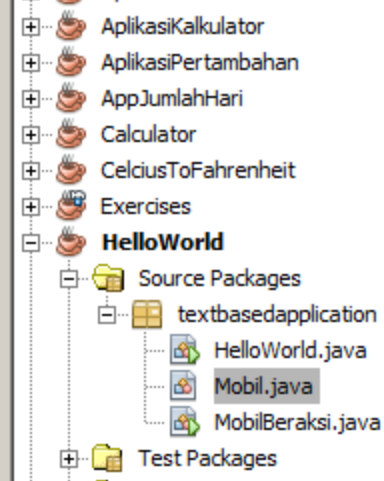


<default config>



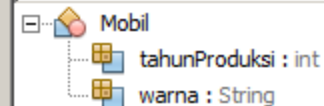
Search (Ctrl+I)

Files Services



Mobil.java - Navigator

Members View



Start Page x HelloWorld.java x Mobil.java x MobilBeraksi.java x



```
1  /*
2      * To change this template, choose Tools | Templates
3      * and open the template in the editor.
4      */
5
6  package textbasedapplication;
7
8  /**
9      *
10     * @author Romi Satria Wahono
11     */
12     public class Mobil {
13         String warna;
14         int tahunProduksi;
15     }
16
```

16:1 INS

Usages

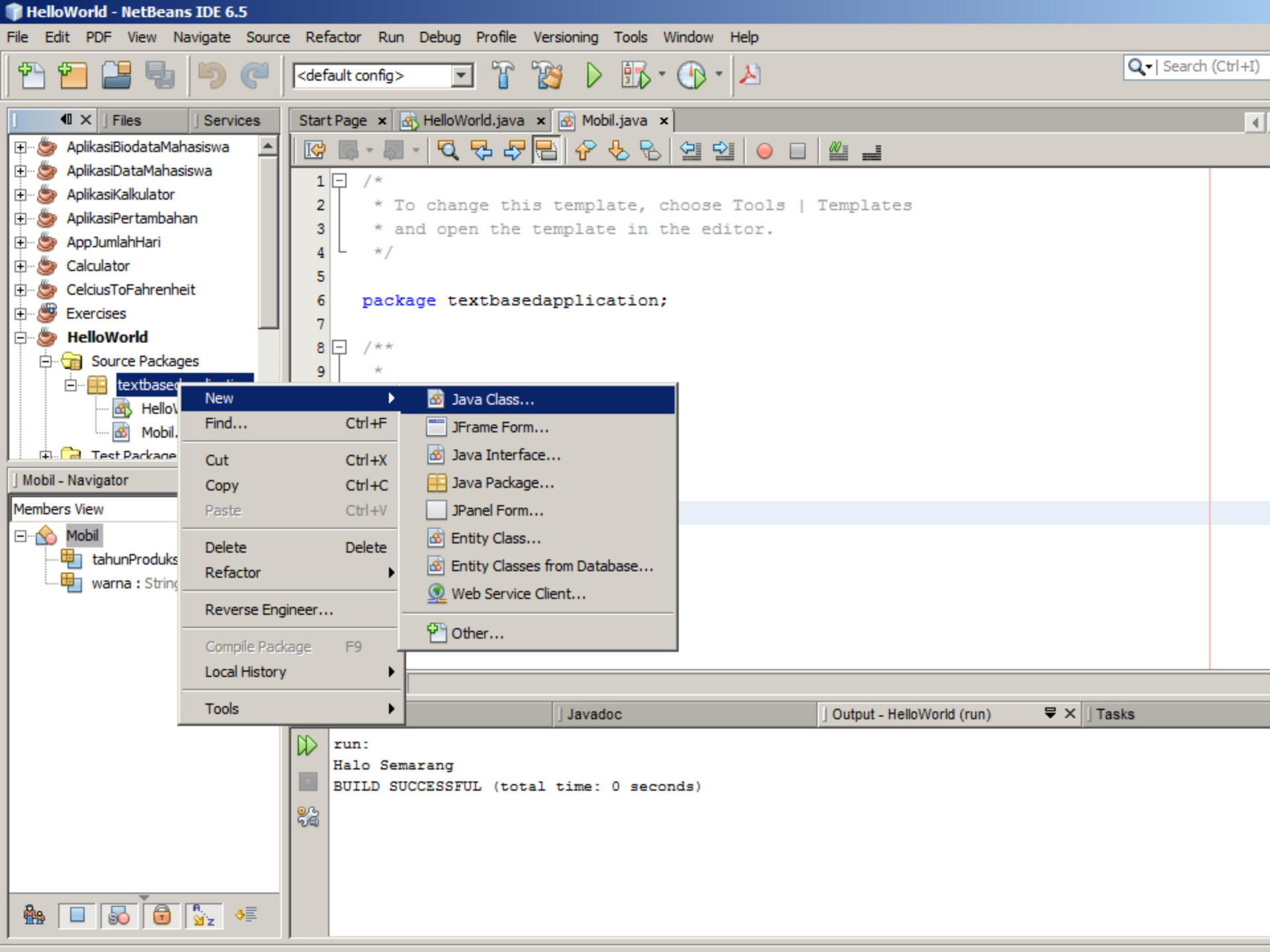
Javadoc

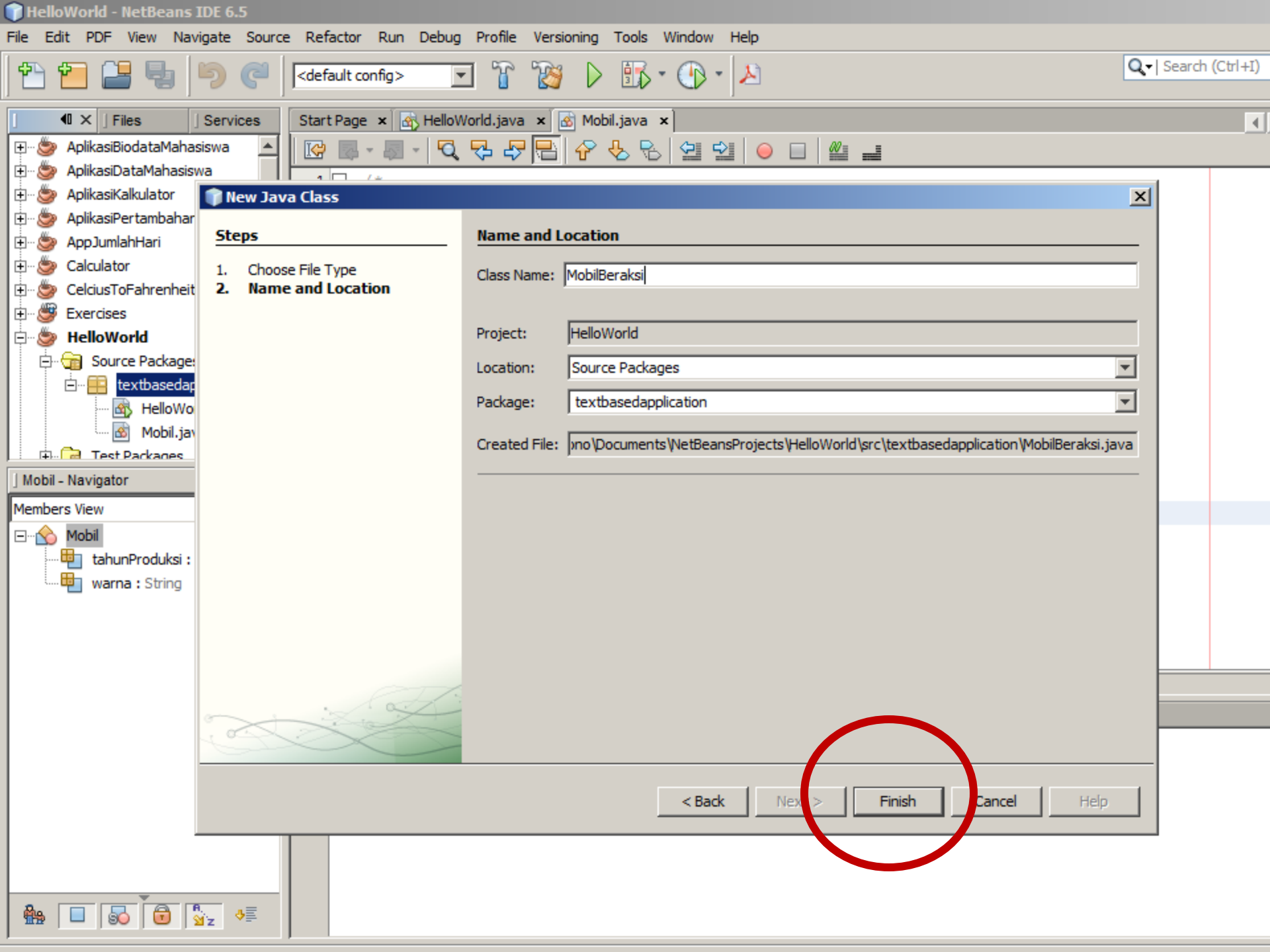
Output - HelloWorld (run)

Tasks

```
run:
Warna: Pink
Tahun Produksi: 1960
BUILD SUCCESSFUL (total time: 0 seconds)
```







NetBeans IDE 6.5

File Edit PDF View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help

<default config> Search (Ctrl+I)

Files Services

- AplikasiBiodataMahasiswa
- AplikasiDataMahasiswa
- AplikasiKalkulator
- AplikasiPertambahan
- AppJumlahHari
- Calculator
- CelsiusToFahrenheit
- Exercises
- HelloWorld**
 - Source Packages
 - textbasedapplication
 - HelloWorld.java
 - Mobil.java
 - MobilBeraksi.java

main - Navigator

Members View

- MobilBeraksi
 - main(String[] args)

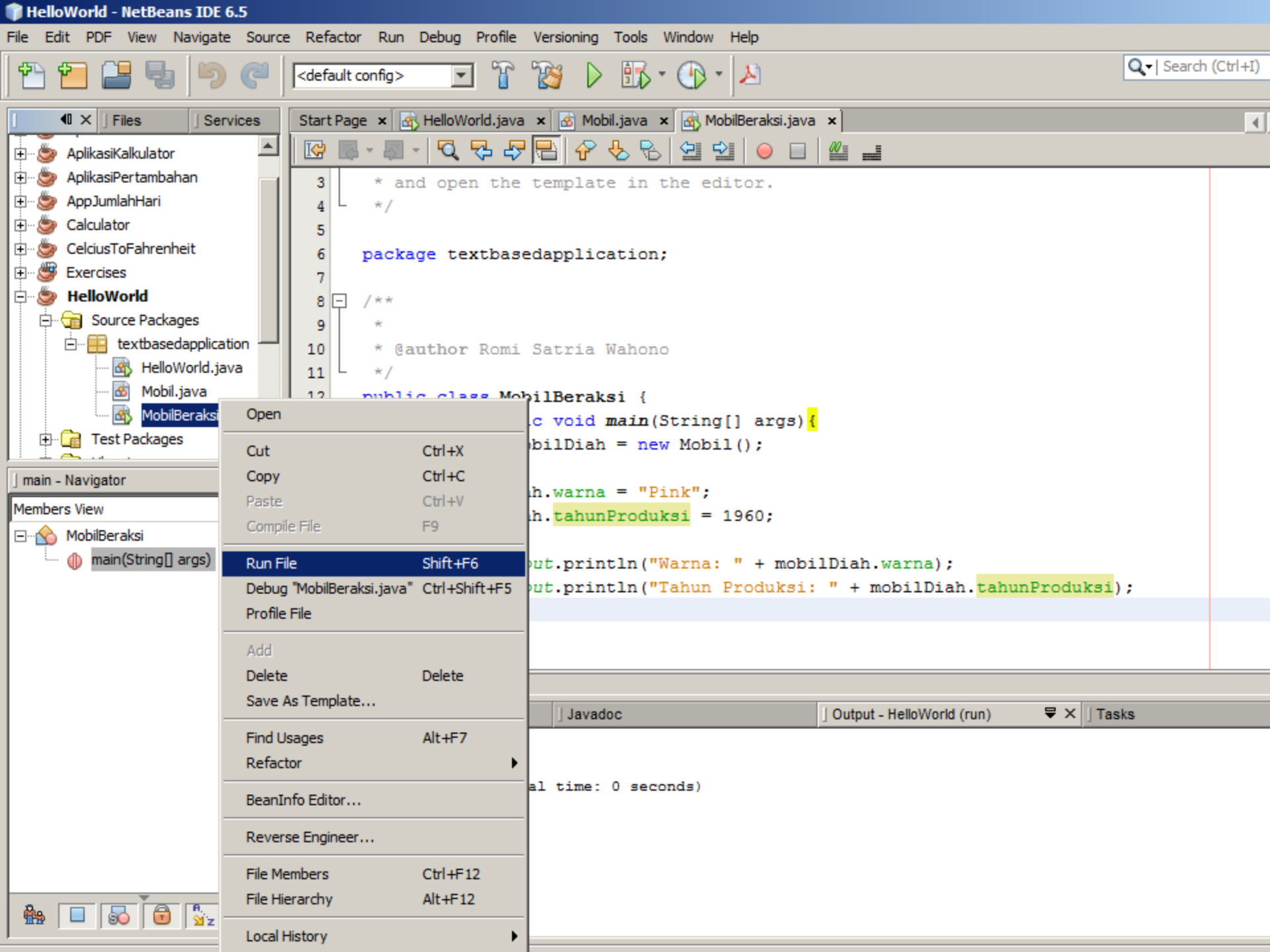
Start Page x HelloWorld.java x Mobil.java x MobilBeraksi.java x

```
3  * and open the template in the editor.
4  */
5
6  package textbasedapplication;
7
8  /**
9   *
10  * @author Romi Satria Wahono
11  */
12  public class MobilBeraksi {
13      public static void main(String[] args) {
14          Mobil mobilDiah = new Mobil();
15
16          mobilDiah.warna = "Pink";
17          mobilDiah.tahunProduksi = 1960;
18
19          System.out.println("Warna: " + mobilDiah.warna);
20          System.out.println("Tahun Produksi: " + mobilDiah.tahunProduksi);
21      }
22  }
```

21:6 INS

Usages Javadoc Output - HelloWorld (run) Tasks

run:
Halo Semarang
BUILD SUCCESSFUL (total time: 0 seconds)



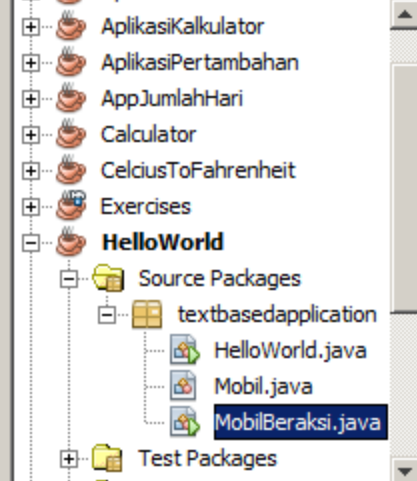


<default config>



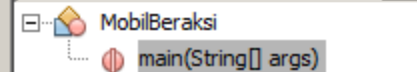
Search (Ctrl+I)

Files Services



main - Navigator

Members View



Start Page x HelloWorld.java x Mobil.java x MobilBeraksi.java x



```
3      * and open the template in the editor.
4      */
5
6      package textbasedapplication;
7
8      /**
9       *
10      * @author Romi Satria Wahono
11      */
12      public class MobilBeraksi {
13          public static void main(String[] args) {
14              Mobil mobilDiah = new Mobil();
15
16              mobilDiah.warna = "Pink";
17              mobilDiah.tahunProduksi = 1960;
18
19              System.out.println("Warna: " + mobilDiah.warna);
20              System.out.println("Tahun Produksi: " + mobilDiah.tahunProduksi);
21          }
22      }
23
```

21:6

Usages

Java

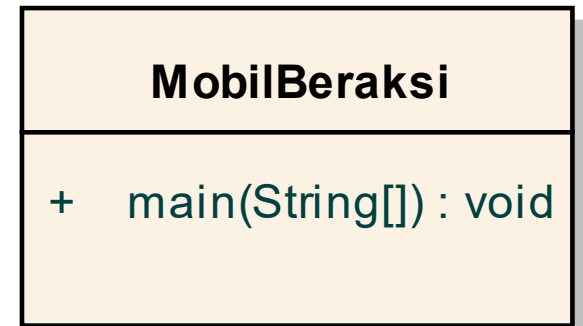
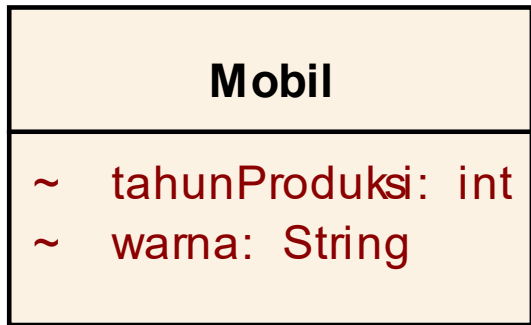
Output - HelloWorld (run)

X

Tasks

```
run:
Warna: Pink
Tahun Produksi: 1960
BUILD SUCCESSFUL (total time: 0 seconds)
```

UML - Class Diagram



Method

- Method adalah implementasi dari bagaimana bekerjanya sebuah class
- Method melakukan:
 1. Manipulasi data
 2. Perhitungan matematika
 3. Memonitor kejadian dari suatu event

Membuat dan Memanggil Method

```
public class Mobil2{
```

```
    String warna;
```

```
    int tahunProduksi;
```

```
    void printMobil(){
```

```
        System.out.println("Warna: " + warna);
```

```
        System.out.println("Tahun: " + tahunProduksi);
```

```
    }
```

```
}
```

Mobil2.java

```
public class Mobil2Beraksi{
```

```
    public static void main(String[] args){
```

```
        // Membuat object
```

```
        Mobil2 mobilku = new Mobil2();
```

```
        /* memanggil atribut dan memberi nilai */
```

```
        mobilku.warna = "Hitam";
```

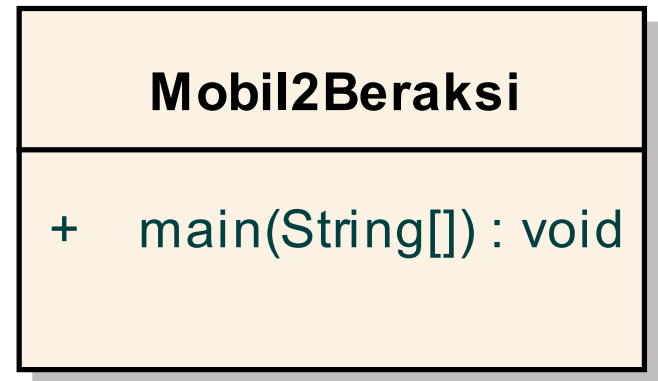
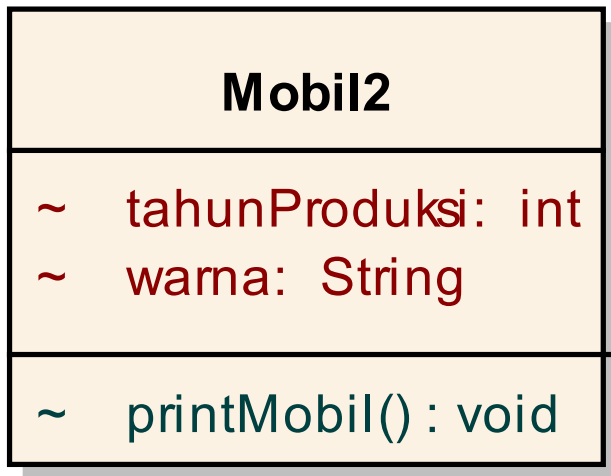
```
        mobilku.tahunProduksi = 2006;
```

```
        mobilku.printMobil();
```

```
    }}
```

Mobil2Beraksi.java

UML - Class Diagram



latihan

- Buat kelas siswa terdiri dari :
- String nama, int tinggi, double berat dan int tahunlahir
- Mempunyai metode cetak()
- Tambahkan metode input(parameter) untuk input data ke atribut dari kelas tersebut
- Buat class nya dan mainnya dengan editplus

■ Buat salah satu sistem di bawah ini :

- -sistem Perpustakaan
- -sistem persewaan mobil
- -sistem persewaan peralatan elektronik
- -sistem penjualan komputer

Buat :

1. Desain diagram UML (use case, sequence, activity dan class diagram dengan menggunakan tools
2. Buat form dari kasus yang dipilih dengan Netbeans, jika tidak bisa akses ke database, boleh formnya ke memory saja atau pakai array
3. Per tim terdiri 3 orang, kumpulkan pertemuan terakhir

Latihan

- Buat class **Handphone** yang berisi **empat method**:
 1. **hidupkan()**
 2. **lakukanPanggilan()**
 3. **kirimSMS()**
 4. **matikan()**
 - Isi masing-masing method dengan tampilan status menggunakan **System.out.println()**
- Buat class **HandphoneBeraksi**, dan panggil method-method diatas dalam class tersebut

Latihan: Hasil Tampilan

Handphone hidup ...

Kring, kring, kring ... panggilan dilakukan

Dung, dung ... sms berhasil terkirim

Handphone mati ...

UML - Class Diagram

MobilLengkap

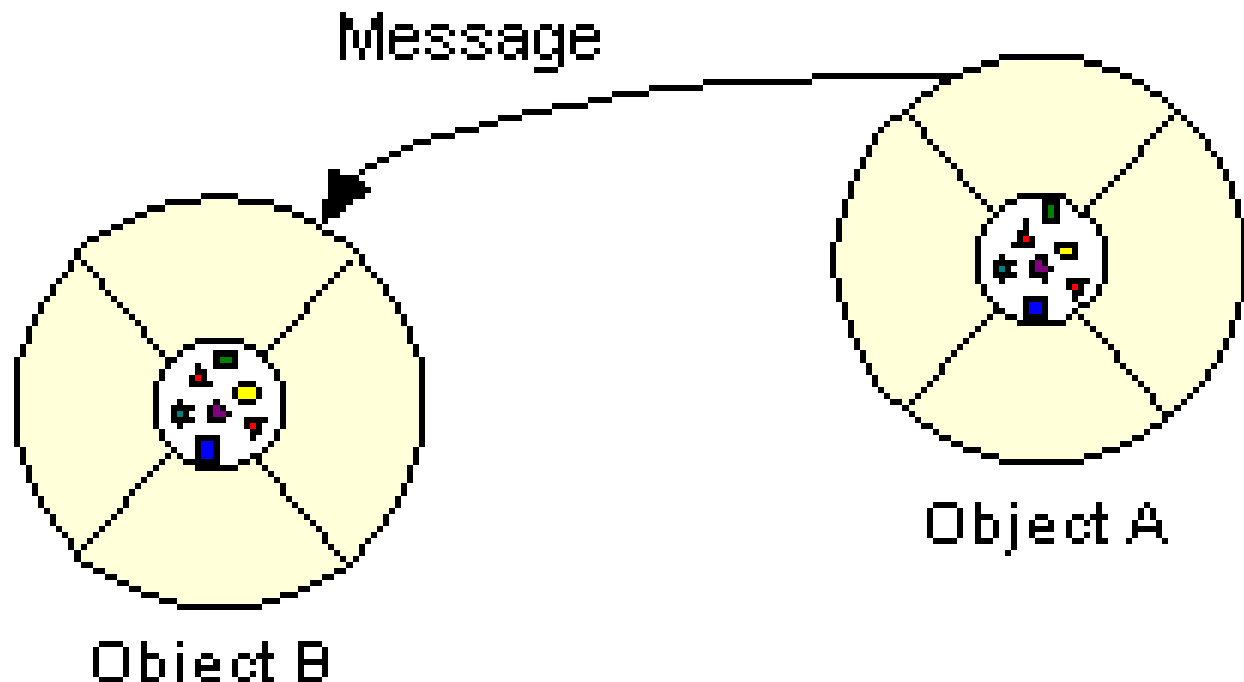
- ~ hidupkanMobil() : void
- ~ matikanMobil() : void
- ~ ubagGigi() : void

MobilLengkapBeraksi

- + main(String[]) : void

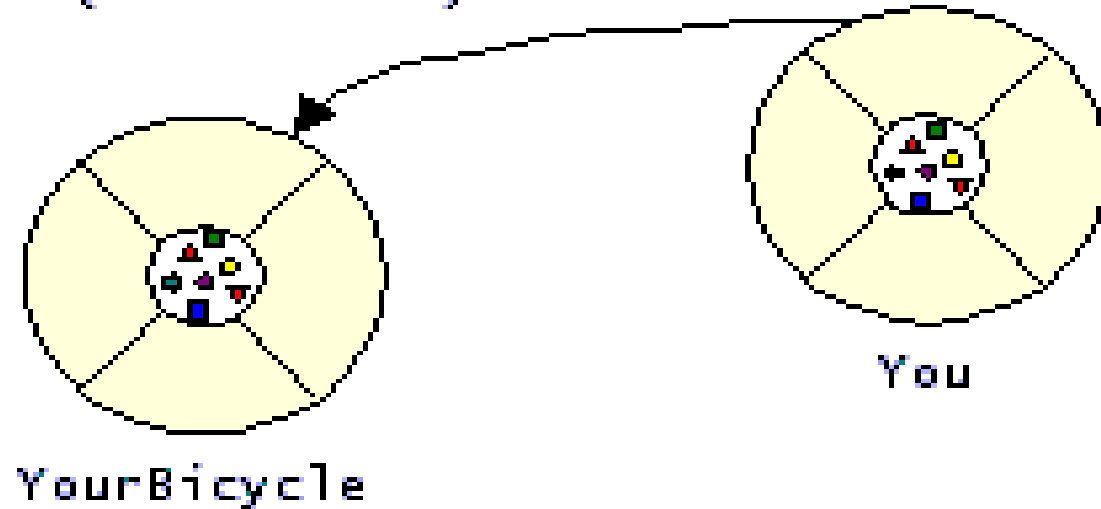
Parameter

- Sepeda (bicycle) akan berguna apabila ada object lain (misalnya anda) yang berinteraksi dengan sepeda (bicycle) tersebut
- Object software berinteraksi dan berkomunikasi dengan object lain dengan cara mengirimkan **message**
- Informasi dalam **message** ini dikenal dengan nama **Parameter**



Pengiriman Pesan dan Parameter

`changeGears(lowerGear)`



1. **You** → object pengirim
2. **YourBicycle** → object penerima
3. **changeGears** → **message** berupa method yang dijalankan
4. **lowerGear** → **parameter** yang dibutuhkan method (message) untuk dijalankan



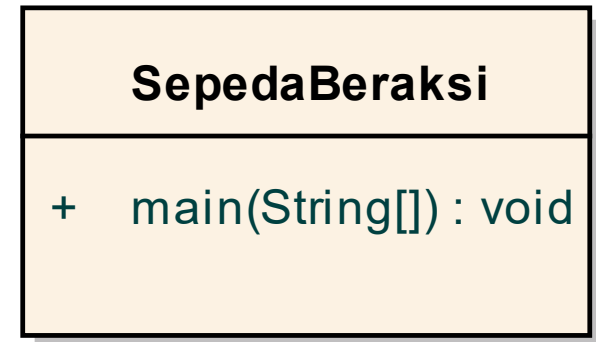
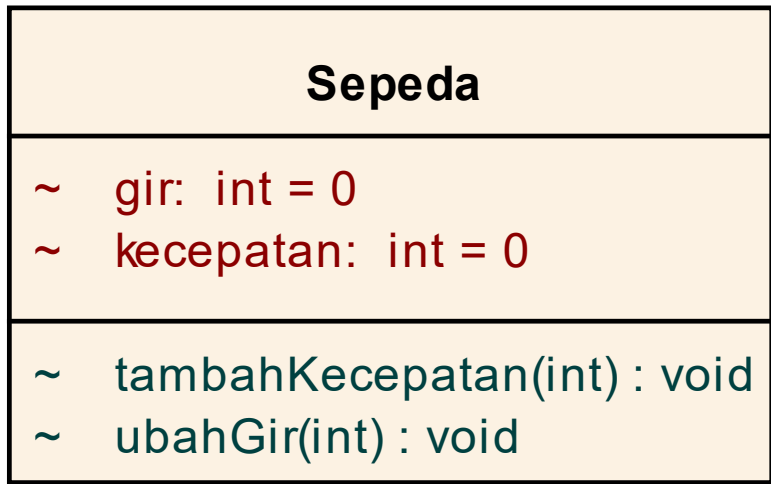
Membuat Class (Sepeda.java)

```
public class Sepeda{  
    int kecepatan, gir;  
  
    // method dengan parameter  
    void ubahGir(int pertambahanGir) {  
        gir= gir+ pertambahanGir;  
        System.out.println(" Gir:" + gir);  
    }  
  
    void tambahKecepatan(int pertambahanKecepatan) {  
        kecepatan = kecepatan+ pertambahanKecepatan;  
        System.out.println(" Kecepatan:" + kecepatan);  
    }  
}
```

Membuat dan Memanggil Object (SepedaBeraksi.java)

```
public class SepedaBeraksi{  
    public static void main(String[] args) {  
        // Membuat object  
        Sepeda sepedaku = new Sepeda();  
  
        /* memanggil atribut dan memberi nilai */  
        sepedaku.kecepatan=10;  
        sepedaku.gir=2;  
  
        // Memanggil method dan menunjuk nilai parameter  
        sepedaku.tambahKecepatan(30);  
        sepedaku.ubahGir(3);  
    }  
}
```

UML - Class Diagram



Latihan

- Buat Class bernama **Matematika**, yang berisi method dengan **dua parameter**:
 - pertambahan(**int a**, **int b**)
 - pengurangan(**int a**, **int b**)
 - perkalian(**int a**, **int b**)
 - pembagian(**int a**, **int b**)
- Buat Class bernama **MatematikaBeraksi**, yang mengeksekusi method dan menampilkan:
 - Pertambahan: $20 + 20 = 40$
 - Pengurangan: $10 - 5 = 5$
 - Perkalian: $10 * 20 = 200$
 - Pembagian: $21 / 2 = 10$

Variasi Tampilan

```
pertambahan(int a, int b){  
    System.out.println(a + " + " + b + " = " + (a+b))  
}
```

```
pertambahan(int a, int b){  
    System.out.println("Hasil = " + (a+b))  
}
```

```
pertambahan(int a, int b){  
    int hasil = a + b;  
    System.out.println("Hasil = " + hasil)  
}
```

Konstruktor -1-

- Method yang digunakan untuk memberi nilai awal pada saat object diciptakan
- Dipanggil secara otomatis ketika **new** digunakan untuk membuat instan class
- Sifat konstruktor:
 - Nama konstruktor sama dengan nama class
 - Tidak memiliki nilai balik dan tidak boleh ada kata kunci void

Konstruktor -2-

```
public class Mobil {
```

```
    String warna;
```

```
    int tahunProduksi;
```

```
    public Mobil(String warna, int tahunProduksi){
```

```
        this.warna = warna;
```

```
        this.tahunProduksi = tahunProduksi;
```

```
    }
```

```
    public void info(){
```

```
        System.out.println("Warna: " + this.warna);
```

```
        System.out.println("Tahun: " + this.tahunProduksi);
```

```
    }
```

Mobil.java

```
public class MobilKonstruktor{
```

```
    public static void main(String[] args){
```

```
        Mobil mobilku = new Mobil("Merah", 2003);
```

```
        mobilku.info();
```

```
    }
```

MobilKonstruktor.java

Kata Kunci this

Digunakan pada pembuatan class dan digunakan untuk **menyatakan object sekarang**

```
public class Mobil{  
    String warna;  
    Int tahunProduksi;  
  
    void isiData(String aWarna,  
                  int aTahunProduksi){  
  
        warna = aWarna;  
        tahunProduksi = aTahunProduksi;  
    }  
}
```

```
public class Mobil{  
    String warna;  
    Int tahunProduksi;  
  
    void isiData(String warna,  
                  int tahunProduksi){  
  
        this.warna = warna;  
        this.tahunProduksi =  
        tahunProduksi;  
    }  
}
```

Latihan

1. Buat class **Buku**

- Tentukan variable buku: **judul, pengarang, penerbit, tahun**
- Buat konstruktor class Buku dengan parameter: **judul, pengarang, penerbit, tahun**
- Buat method **cetakBuku**

2. Buat class **BukuBeraksi** dan tampilkan 2 buku:

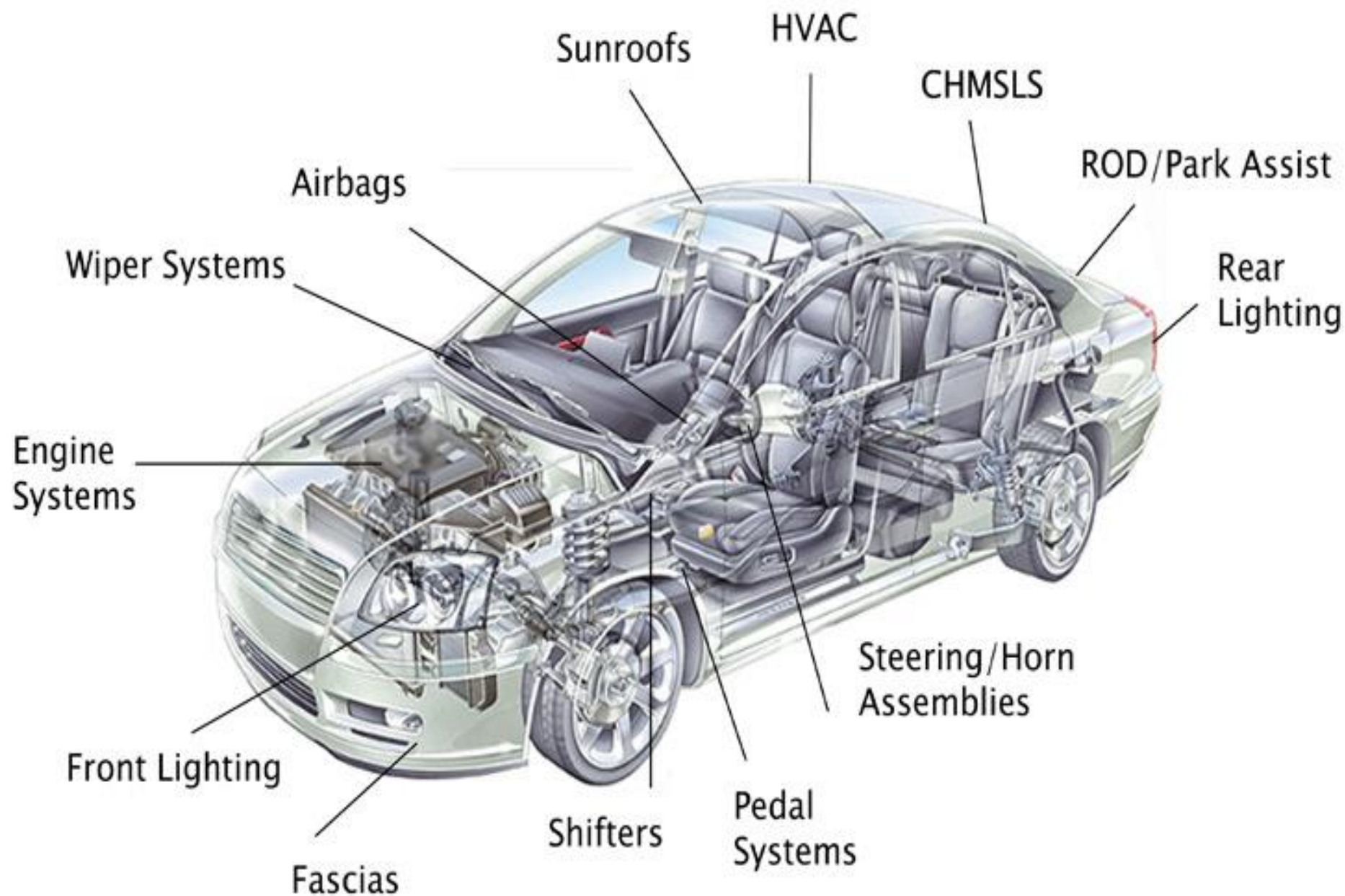
- Rogers Cadenhead & Laura Lemay, Teach Yourself Java 6 in 21 Days, Sams Publishing, 2007
- Deitel & Deitel, Java How to Program 7th Edition, Prentice Hall, 2007

Karakteristik Pemrograman Berorientasi Objek

Abstraction, Encapsulation,
Inheritance, Polymorphism

Abstraction

- Cara kita melihat suatu sistem dalam **bentuk yang lebih sederhana**, yaitu sebagai suatu kumpulan subsistem (object) yang saling berinteraksi.
 - Mobil adalah kumpulan sistem pengapian, sistem kemudi, sistem pengereman
- Alat meng-abstraksikan sesuatu adalah **class**
- Object bersifat **modularity**. Object dapat ditulis dan **dimaintain terpisah (independen)** dari object lain



BICYCLE

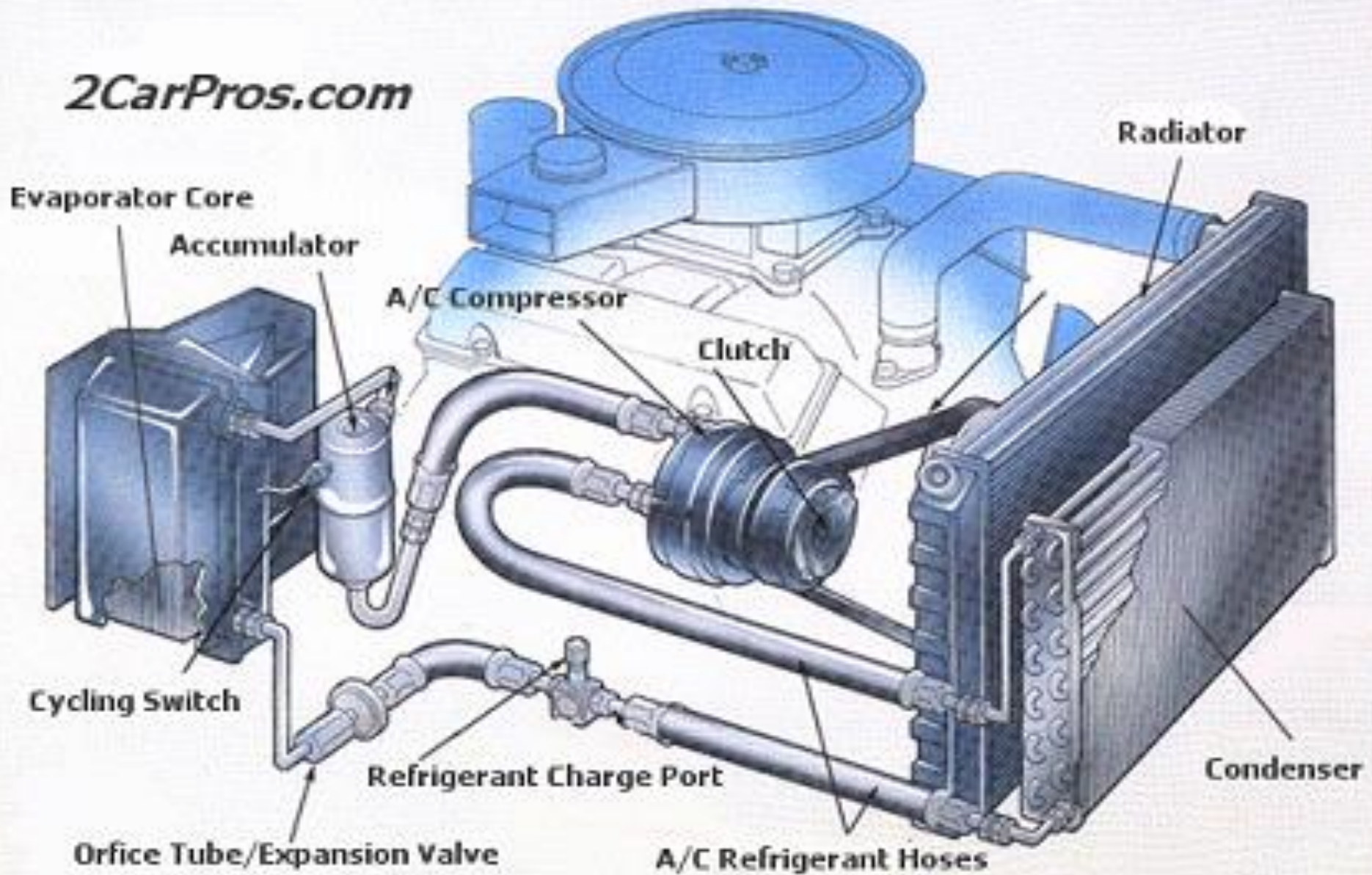


Encapsulation

- Mekanisme **menyembunyikan suatu proses dalam sistem** untuk menghindari interferensi dan menyederhanakan penggunaan sistem itu sendiri
 - Tongkat transmisi (gigi) pada mobil
 - Tombol on/off/pengaturan suhu pada AC
- Class access level (public, protected, privat) adalah **implementasi dari konsep encapsulation**

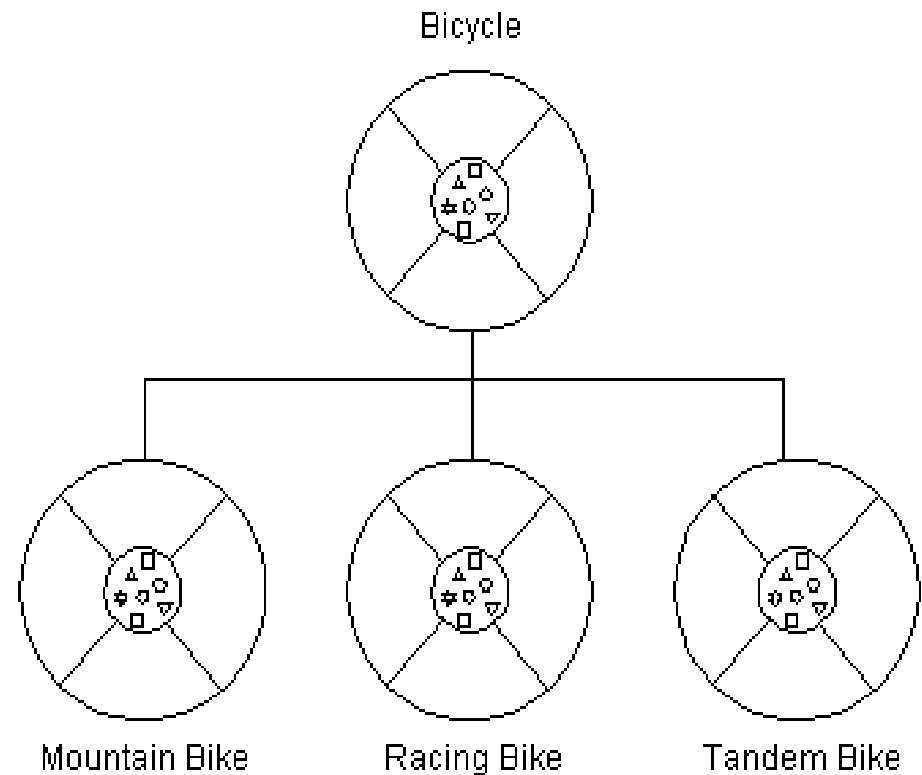






Inheritance (Pewarisan)

- Suatu class dapat **mewariskan atribut dan method** kepada class lain (subclass), serta membentuk class hierarchy
- Penting untuk **Reusability**
- Java Keyword: ***extends***



Sepeda.java

```
public class Sepeda{  
    int kecepatan, gir ;  
  
    // method  
    void ubahGir(int pertambahanGir) {  
        gir= gir+ pertambahanGir;  
        System.out.println("Gir:" + gir);  
    }  
  
    void tambahKecepatan(int pertambahanKecepatan) {  
        kecepatan = kecepatan+ pertambahanKecepatan;  
        System.out.println("Kecepatan:" + kecepatan);  
    }  
}
```

Class SepedaGunung Mewarisi Class Sepeda

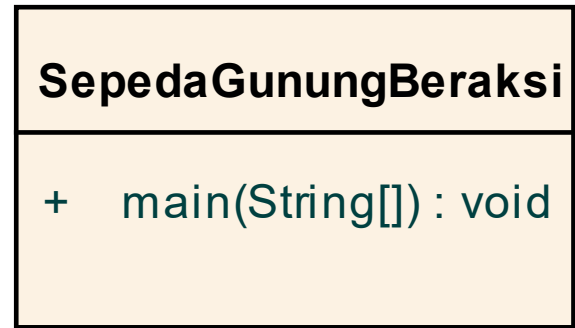
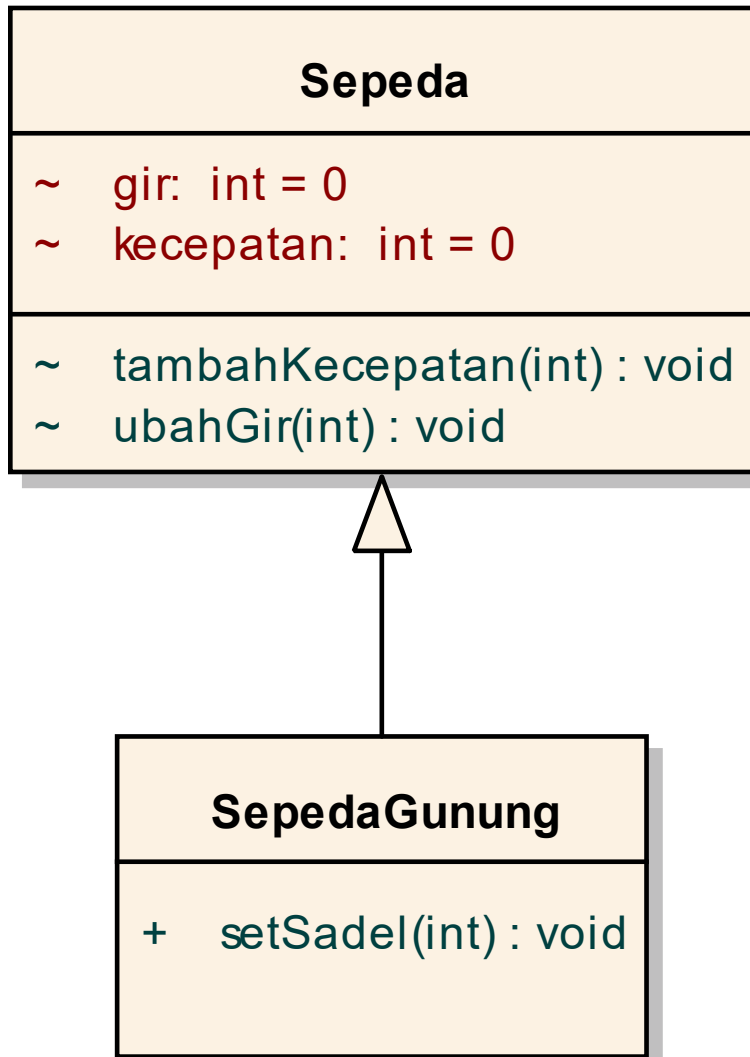
```
class SepedaGunung extends Sepeda{  
  
    public void setSadel(int nilaiSadel) {  
        System.out.println("Tinggi Sadel:" +  
            nilaiSadel);  
    }  
}
```

SepedaGunung.java

```
class SepedaGunungBeraksi {  
    public static void main(String[] args) {  
        // Membuat object  
        SepedaGunung spd= new  
            SepedaGunung();  
  
        // Memanggil method di object  
        spd.tambahKecepatan(10);  
        spd.ubahGir(2);  
        spd.setSadel(20);  
  
    }  
}
```

SepedaGunungBeraksi.java

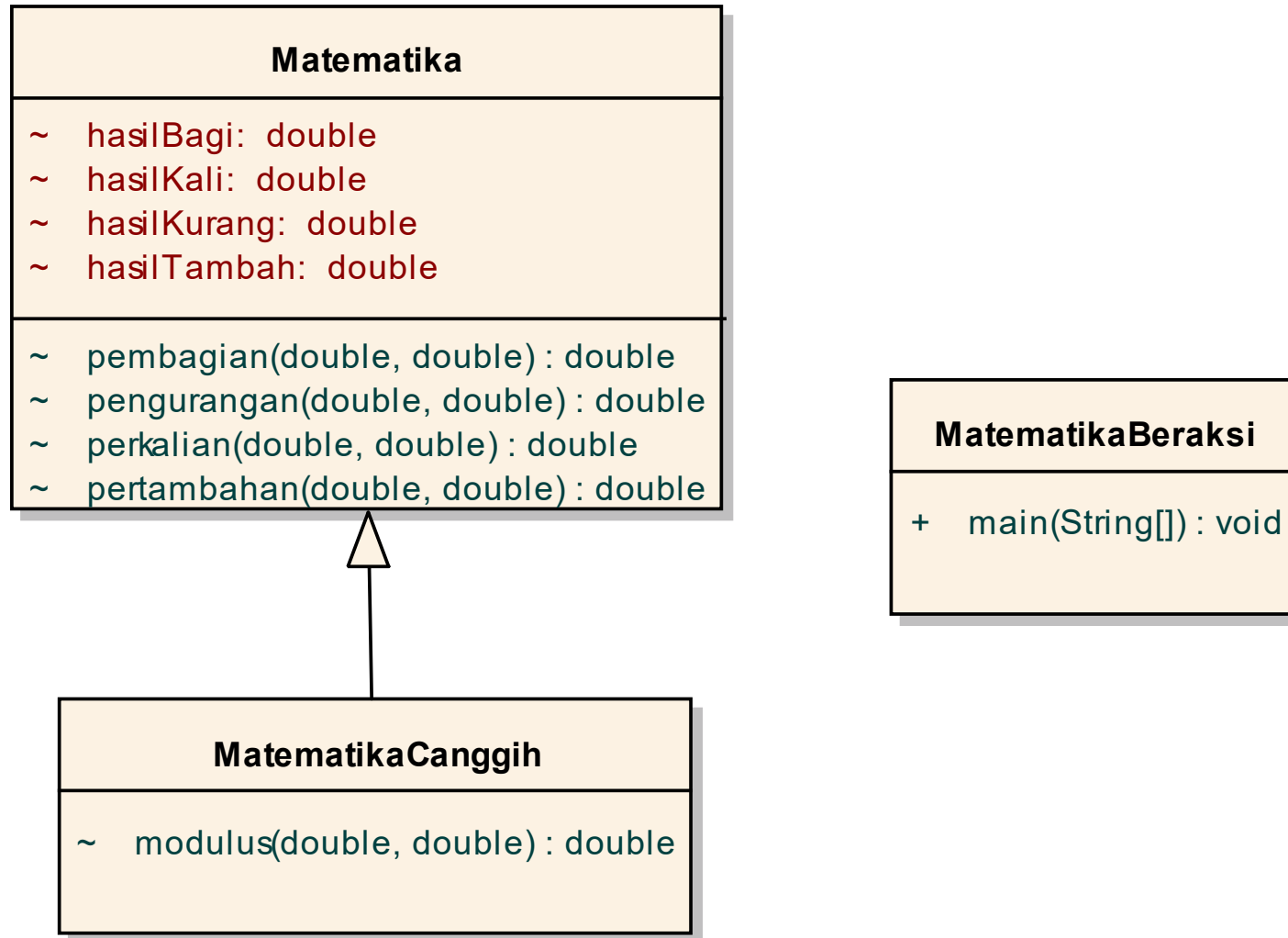
UML – Class Diagram



Latihan

- Buat class **MatematikaCanggih** yang merupakan **inherit** dari class **Matematika**
 - Tambahkan method **modulus(int a, int b)** yang menghitung modulus dari a dan b. Operator modulus adalah **%**
- Buat class **MatematikaCanggihBeraksi** yang memanggil method pertambahan, perkalian dan modulus

UML – Class Diagram



Polymorphism

- Suatu object dapat memiliki berbagai bentuk implementasi dari metoda
- Implementasi konsep polymorphism:
 1. **Overloading**: Penggunaan **satu nama** untuk beberapa **method yang berbeda parameter**
 2. **Overriding**: Terjadi ketika deklarasi method subclass persis **sama dengan** method dari superclassnya

Konstruktor adalah metoda yang namanya sama dengan nama kelas, dipanggil saat instansiasi sehingga atribut bisa diisi data via parameter

Jika konstruktor lebih dari 1 disebut overloadng konstruktor

Polymorphism – Overloading

```
class Mobil {  
    String warna;  
    int tahunProduksi;  
  
    public Mobil(String warna, int tahunProduksi){  
        this.warna = warna;  
        this.tahunProduksi = tahunProduksi;  
    }  
  
    public Mobil(){  
    }  
  
    void info(){  
        System.out.println("Warna: " +  
            this.warna);  
        System.out.println("Tahun: " +  
            this.tahunProduksi);  
    }  
}
```

```
public class MobilKonstruktor{  
    public static void main(String[] args){  
        Mobil mobilku = new Mobil("Merah",  
            2003);  
        mobilku.info();  
  
        Mobil mobilmu = new Mobil();  
        mobilmu.info();  
    }  
}
```

C:\Windows\system32\cmd.exe

```
Warna: Merah  
Tahun: 2003  
Warna: null  
Tahun: 0  
Press any key to continue . . . _
```

Polymorphism – Overloading

```
class Lingkaran{  
    public buatLingkaran(){  
    }  
    public buatLingkaran(int diameter){  
    ...  
    }  
    public buatLingkaran(int diameter, int x, int y){  
    ...  
    }  
    public buatLingkaran(int diameter, int x, int y, int warna, String  
    namaLingkaran){  
    ...  
    }  
}
```

Polymorphism - Overriding

```
class Sepeda{  
    int kecepatan, gir ;  
  
    // method  
    void ubahGir(int pertambahanGir) {  
        gir= gir+ pertambahanGir;  
        System.out.println("Gir:" + gir);  
    }  
  
    void tambahKecepatan(int pertambahanKecepatan) {  
        kecepatan = kecepatan+ pertambahanKecepatan;  
        System.out.println("Kecepatan:" + kecepatan);  
    }  
}
```

Class SepedaGunung Mewarisi Class Sepeda

```
class SepedaGunung extends Sepeda {  
  
    void ubahGir(int pertambahanGir) {  
        gir= 2*(gir+ pertambahanGir );  
        System.out.println("Gir:" + gir);  
    }  
}
```

SepedaGunung.java

```
class SepedaGunungBeraksi {  
    public static void main(String[] args) {  
        // Membuat object  
        SepedaGunung sepedaku = new SepedaGunung();  
  
        // Memanggil method di object  
        sepedaku.tambahKecepatan(10);  
        sepedaku.ubahGir(2);  
  
    }  
}
```

SepedaGunungBeraksi.java

Latihan

1. Kembangkan class **Matematika** dan **MatematikaBeraksi**
2. Lakukan **overloading** pada **Method** yang ada (pertambahan, pengurangan, perkalian, pembagian)
3. Tambahkan method baru bertipe data **double** (pecahan) dan **memiliki 3 parameter**
4. Uji di kelas **MatematikaBeraksi** dengan parameter pecahan: 12.5, 28.7, 14.2
5. Misalnya:
pertambahan(12.5, 28.7, 14.2)
pertambahan(23, 34)

Tugas Besar

- Buat kelas diagram : super kelas karyawan dg attrib string nama, int umur, double berat, metode Hitunggaji();
- Buat kelas karyawantetap dg attrib double gapok, double tunjangan, metode Hitunggaji() dg return value totalgaji
- Buat kelas karyawankontrak dg attrib double honorperhari, int jumlahharimasuk dg metode Hitunggaji() dg return value totalupah
- Kelas karyawantetap dan karyawankontrak sub kelas karyawan
- Implementasikan class diagram dengan tools (EA, Visual paradigm, argoUML, PowerDesigner, starUML atau yang lain (misal Netbean UML plug-in)
- Generate (File Export) menjadi coding Java
- Kembangkan kelas menjadi 8-10 kelas atau lebih dengan mengcover kelas pola B,C,E (Boundary, Control dan Entity class)
- Lengkapi dengan Use Case Diagram, Sequence Diagram dan Activity Diagram dengan tools tsb
- Lengkapi dengan GUI dalam bahasa pemrograman tertentu untuk 1-2 Boundary Class
- Dikerjakan berkelompok maks 2 orang diserahkan minggu terakhir (print out)

Matematika.java

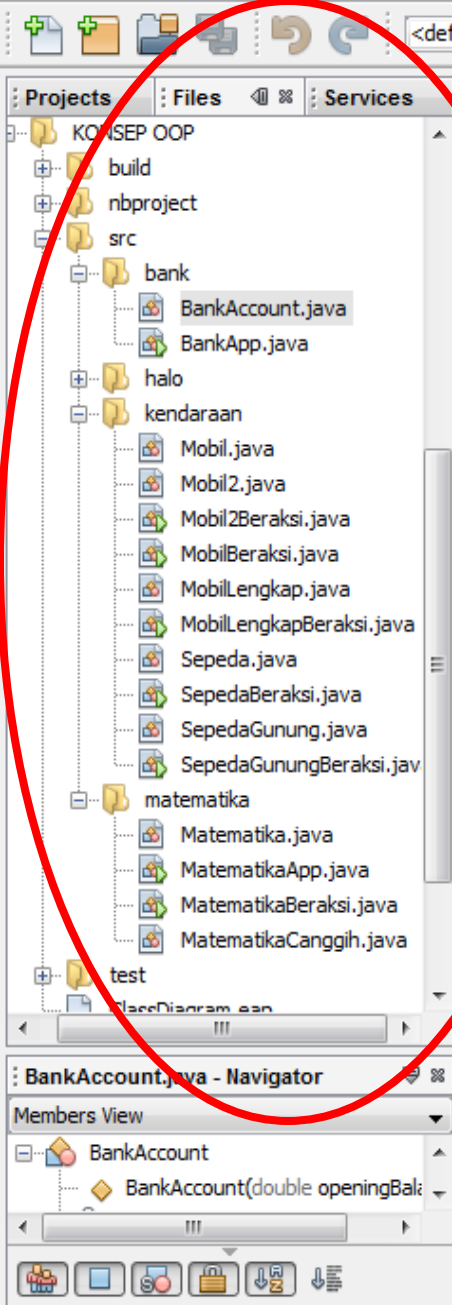
```
class Matematika{  
    void pertambahan (int a, int b){  
        int hasil= a + b;  
        System.out.println("hasil:" + hasil);  
    }  
  
    void pertambahan (double a, double b, double c){  
        int hasil= a + b + c;  
        System.out.println("hasil:" + hasil);  
    }  
    ...  
}
```

Pengorganisasian Class

Package, Interface

Packages

- Package adalah koleksi dari beberapa class dan interface yang berhubungan, dan menyediakan proteksi akses dan pengelolaan namespace
- 1 package adalah 1 folder di file system
- Package berguna untuk mengorganisir file dalam suatu project atau library
- Nama package menggunakan lowercase
- Nama package mengikuti nama domain (perusahaan) dengan susunan terbalik
 - Contoh: com.brainmatics.kendaraan
- Keyword: *package name;*



The screenshot shows the NetBeans IDE interface with the following components:

- Projects, Files, Services:** The top-left pane showing the project structure. The 'src' folder is expanded, showing subfolders like 'bank', 'halo', and 'kendaraan', each containing Java files.
- Project Explorer:** The middle-left pane showing the 'BankAccount.java' file selected in the 'bank' folder.
- BankAccount.java - Navigator:** The bottom-left pane showing the 'BankAccount' class and its members, including the 'BankAccount(double openingBal)' constructor.
- Source Editor:** The main workspace showing the code for 'BankAccount.java'. The code includes package declarations, class definitions, and methods.
- Output - KONSEP OOP (run):** The bottom-right pane showing the execution output, which includes an exception message and a successful build status.

```
1  /*
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5
6  package bank;
7
8  /**
9   *
10  * @author ROMI SATRIA WAHONO
11  */
12  public class BankAccount {
13
14      private double balance;
15      String test;
16
17      public BankAccount(double openingBalance) {
18          balance = openingBalance;
19          System.out.println("Balance awal sebelum transaksi = " + openingBalance);
20      }
21
22      public void deposit(double amount) {
```

run:
Exception in thread "main" java.lang.RuntimeException: Uncompilable source code
at kendaraan.SepedaGunungBeraksi.main(SepedaGunungBeraksi.java:15)
Java Result: 1
BUILD SUCCESSFUL (total time: 2 seconds)

Budi.java

```
package kelasku;
```

```
public class Budi{
```

```
    public void info(){
```

```
        System.out.println("Kelas Budi");
```

```
    }
```

```
}
```

Joko.java

```
package kelasku;
```

```
public class Joko{
```

```
    public void info(){
```

```
        System.out.println("Kelas Joko");
```

```
    }
```

```
}
```

PaketBeraksi.java

```
import kelasku.*;
```

```
public class PaketBeraksi{  
    public static void main(String[] args){  
        Budi objectBudi = new Budi();  
        objectBudi.info();  
        Joko objectJoko = new Joko();  
        objectJoko.info();  
    }  
}
```

Struktur Direktori

- PaketBeraksi.java
- **kelasku**
 - Budi.java
 - Joko.java

Interface

- Interface digunakan apabila kita ingin menentukan apa yang harus dilakukan oleh suatu class tapi **tidak menentukan bagaimana cara untuk melakukannya**
- Interface sebenarnya sama dengan class, tapi hanya memiliki **deklarasi method tanpa implementasi**

InterfaceLampu.java

```
interface InterfaceLampu{  
    public static final int KeadaanHidup=1;  
    public static final int KeadaanMati=0;  
  
    public abstract void hidupkan();  
    public abstract void matikan();  
}
```

Lampu.java

```
public class Lampu implements InterfaceLampu{
    int statusLampu;

    public void hidupkan(){
        if (this.statusLampu == KeadaanMati){
            this.statusLampu = KeadaanHidup;
            System.out.println("Hidupkan Lampu! --> Lampu Hidup");
        }else{
            System.out.println("Hidupkan Lampu! --> Lampu Sudah Hidup Kok");
        }
    }

    public void matikan(){
        if (this.statusLampu == KeadaanHidup){
            this.statusLampu = KeadaanMati;
            System.out.println("Matikan Lampu! --> Lampu Mati");
        }else{
            System.out.println("Matikan Lampu! --> Lampu Sudah Mati Kok");
        }
    }
}
```

LampuBeraksi.java

```
public class LampuBeraksi{  
    public static void main(String[] args){  
  
        Lampu lampuKamar = new Lampu();  
  
        System.out.println("Status Lampu Saat Ini: Mati");  
  
        lampuKamar.hidupkan(); //Hidupkan Lampu  
        lampuKamar.matikan(); //Matikan Lampu  
        lampuKamar.matikan(); //Matikan Lampu  
        lampuKamar.hidupkan(); //Hidupkan Lampu  
        lampuKamar.hidupkan(); //Hidupkan Lampu  
    }  
}
```

Files

- Test Libraries
- OOP Concepts
 - Source Packages
 - buku
 - Buku.java
 - BukuApp.java
 - halo
 - HaloJakarta.java
 - kendaraan
 - Mobil.java
 - Mobil2.java
 - Mobil2App.java
 - MobilApp.java
 - lampu
 - InterfaceLampu.java
 - Lampu.java
 - LampuBeraksi.java
 - matematika

```

10  * @author ROMI SAIKIA WAHONO
11  */
12  public class LampuBeraksi {
13      public static void main(String[] args) {
14
15          Lampu lampuKamar = new Lampu();
16
17          System.out.println("Status Lampu Saat Ini: Mati");
18
19          lampuKamar.hidupkan(); //Hidupkan Lampu
20          lampuKamar.matikan(); //Matikan Lampu
21          lampuKamar.matikan(); //Matikan Lampu
22          lampuKamar.hidupkan(); //Hidupkan Lampu
    
```

main - Navigator

Members View

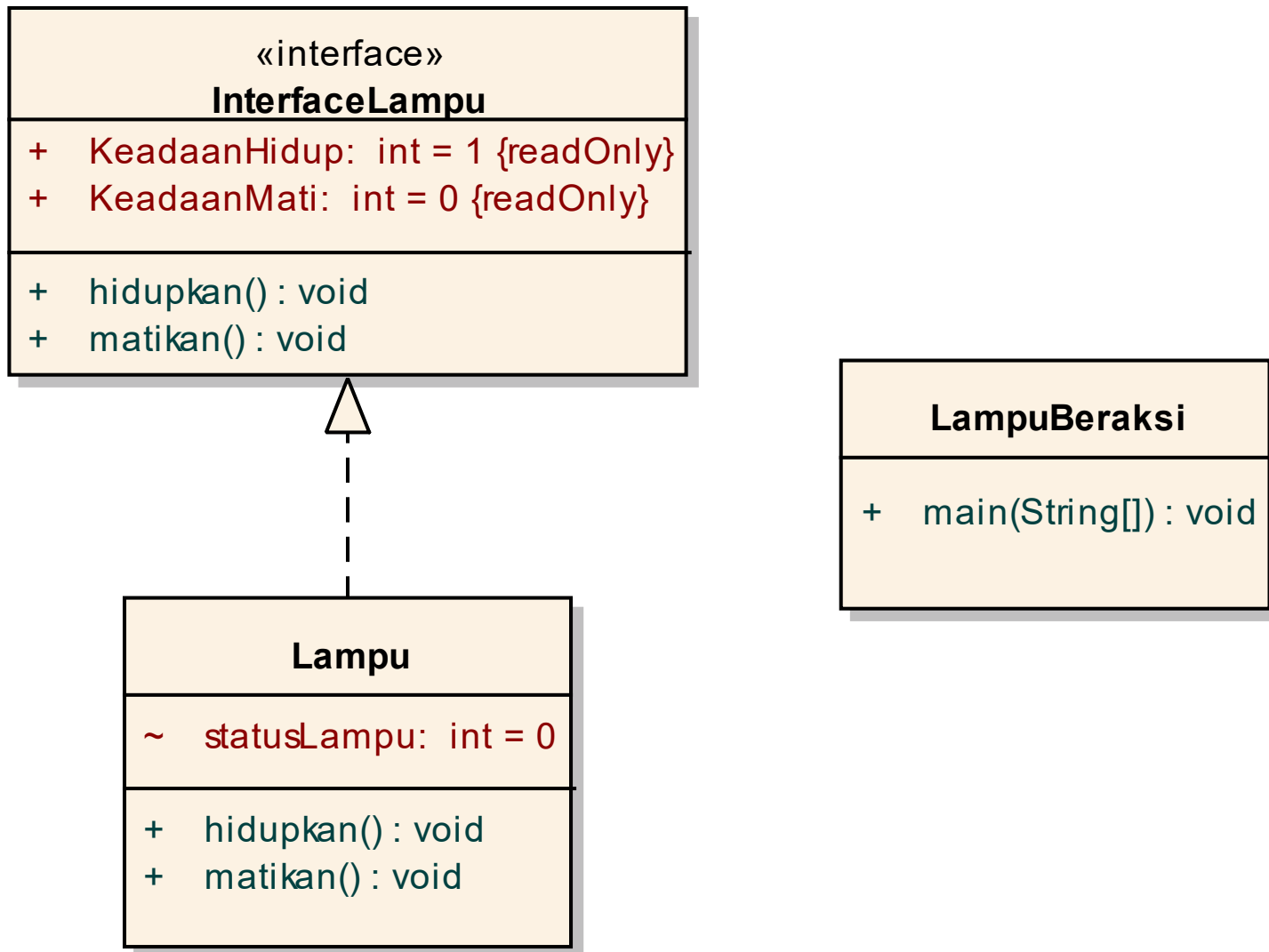
- main(String[] args)

Output - OOP Concepts (...)

```

run:
Status Lampu Saat Ini: Mati
Hidupkan Lampu! --> Lampu Hidup
Matikan Lampu! --> Lampu Mati
Matikan Lampu! --> Lampu Sudah Mati Kok
Hidupkan Lampu! --> Lampu Hidup
Hidupkan Lampu! --> Lampu Sudah Hidup Kok
BUILD SUCCESSFUL (total time: 0 seconds)
    
```

UML – Class Diagram



Latihan

- Buat program serupa dengan InterfaceBeraksi.java tapi untuk **ACBeraksi.java**
- Masukkan method-method di bawah ke dalam **ACBeraksi.java**
 - matikanAC() dan hidupkanAC()
 - dinginkanAC() dan panaskanAC ()

Summary -1-

- **Software engineer** is a key person for **analyzing** the business, **identifying** opportunities for improvement, **designing** and **implement** the software systems
- **Software Development Lifecycle (SDLC)** consists of four stages:
 1. Planning
 2. Analysis
 3. Design
 4. Implementation

Summary -2-

- The major software development methodologies:

1. Structured design

- ✓ Waterfall method
- ✓ Parallel development

2. Rapid Application Development

- ✓ Phased Development
- ✓ Prototyping
- ✓ Throw-away Prototyping

3. Agile development

- ✓ Extreme Programming (XP)

Summary -3-

- UML is the **standard language** for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system
- UML version 2.0 has **14 diagrams in 2 groups**:
 - Structure Diagrams
 - Behavior Diagrams
- Most used UML diagrams: **Use Cases** Diagram, **Sequence** Diagram, **Class** Diagram, **Activity** Diagrams, **Package** Diagrams, **Deployment** Diagrams
- Object-Oriented Paradigm

Referensi (Foundation)

- Roger S. Pressman, **Software Engineering: A Practitioner's Approach Seventh Edition**, McGraw-Hill, 2009
- Ian Sommerville, **Software Engineering 9th Edition**, Addison-Wesley, 2010
- Albert Endres dan Dieter Rombach, **A Handbook of Software and Systems Engineering**, Pearson Education Limited, 2003
- Yingxu Wang, **Software Engineering Foundations: A Software Science Perspective**, Auerbach Publications, Taylor & Francis Group, 2008
- **Guide to the Software Engineering Body of Knowledge 2004 Version (SWEBOK)**, IEEE Computer Society, <http://www.swebok.org>, 2004

Referensi (Process)

- Alan Dennis et al, **Systems Analysis and Design with UML – 3rd Edition**, John Wiley and Sons, 2010
- Dan Pilone and Russ Miles, **Head First Software Development**, O'Reilly Media, 2008
- Barclay and Savage, **Object-Oriented Design with UML and Java**, Elsevier, 2004
- Paul Kimmel, **UML Demystified**, McGraw-Hill, 2005
- Kim Hamilton and Russell Miles, **Learning UML 2.0**, O'Reilly, 2006
- Howard Podeswa, **UML for the IT Business Analyst**, Course Technology, 2009
- Deloitte, **Business Process Modeling – Basic Guideline and Tips**, 2008

Referensi (Quality Assurance)

- Daniel Galin, **Software Quality Assurance**, Addison-Wesley, 2004
- Jeff Tian, **Software Quality Engineering**, John Wiley & Sons, Inc., 2005
- G. Gordon Schulmeyer, **Handbook of Software Quality Assurance Fourth Edition**, Artech House, 2008
- Kshirasagar Naik and Priyadarshi Tripathy, **Software Testing and Quality Assurance**, John Wiley & Sons, Inc., 2008