CE204 Object-Oriented Programming

Software Engineering and OOP with Java -I

Author: Asst. Prof. Dr. Uğur CORUH

Contents

List of Figures

List of Tables

0.1 CE204 Object-Oriented Programming

0.2 Week-1 (Software Engineering and OOP with Java -I)

0.2.0.1 Spring Semester, 2021-2022

Download DOC[1], SLIDE[2], PPTX[3]

0.3 Brief Description of Course and Rules

We will first talk about,

1. Course Plan and Communication
2. Grading System, Homeworks, and Exams

please read the syllabus carefully.

0.4 Course Growth Track

- OOP with Java
- UML
- PlantUML + UMPLE + UML
- PlantUML + UMPLE + UML + Java
- Design Patterns + UML + Java + UMPLE

0.5 Outline (1)

- Software and Software Engineering
- Object Orientation
- Basing Software Development on Reusable Technology

0.6 Outline (2)

- OOP with Java Intro
- Java Classes
- Java Objects
- Java Methods
- Java Inheritance
- Java Access Modifiers
- This and InstanceOf Keywords

0.7 Software and Software Engineering

0.8 Software

- Computer Software is the product that software engineers design and build.
- It encompasses

[1] ce204-week-1.md_doc.pdf
[2] ce204-week-1.md_slide.pdf
[3] ce204-week-1.md_slide.pptx

  - **programs** that execute within a computer of any size and architecture,
  - **documents** that encompass hard-copy and virtual forms,
  - **data** that combine numbers and text but also includes representations of pictorial, video, and audio information.

0.9 The Nature of Software... (1)

- **Software is intangible** Hard to understand development effort
- **Software is easy to reproduce**
  - Cost is in its development
    - in other engineering products, manufacturing is the costly stage
- **The industry is labor-intensive**
  - Hard to automate
- **Untrained people can hack something together**
  - Quality problems are hard to notice

0.10 The Nature of Software... (2)

- **Software is easy to modify**
  - People make changes without fully understanding it
- **Software does not "wear out"**
  - It deteriorates by having its design changed:
    - erroneously, or
    - in ways that were not anticipated, thus making it complex
- **Conclusions**
  - Much software has poor design and is getting worse
  - We have to learn to 'engineer' software

0.11 Some types of Software

- **Real time embedded software**
  - E.g. control and monitoring systems
  - Must react immediately
  - Safety often a concern
- **Data processing software**
  - Used to run businesses
  - Accuracy and security of data are key
- **Game software**
- **Mobile device software**
- **Web-based software**
- *Etc.*

0.12 Single Chance

Most of the clients use applications to ease their tasks to make money. For this reason, you do not have a chance to try your application development in real system. Before this deployment you have to use

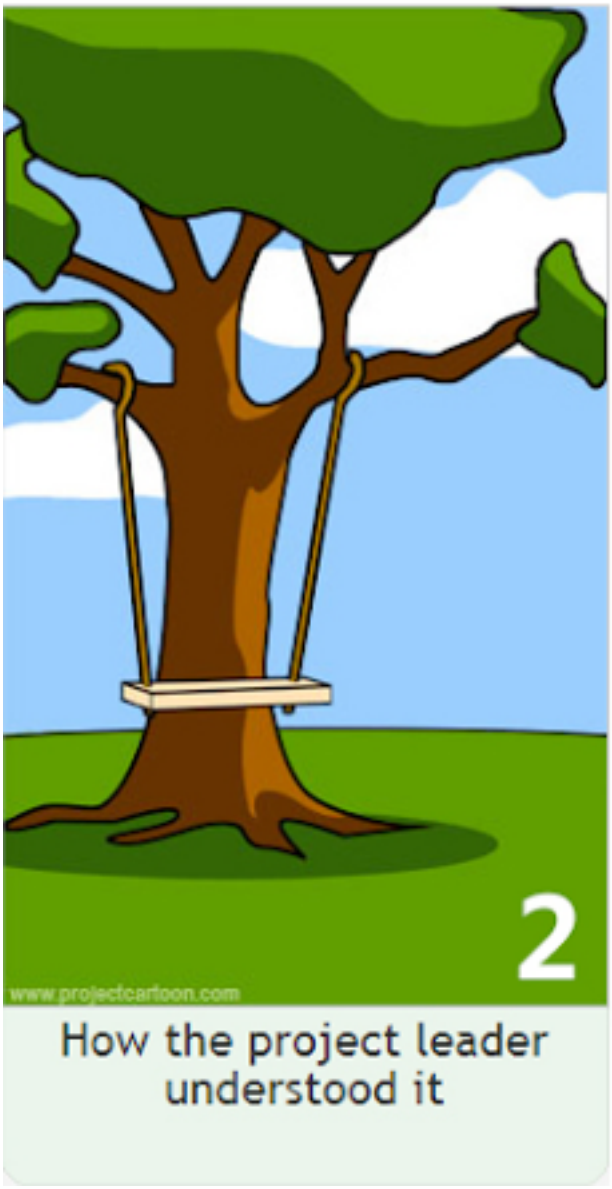
- Theory
- Experimentation
- Guesses
- Feedback

Lets talk about.

0.13 The Project Construction Cycle - The Tree Swing Example

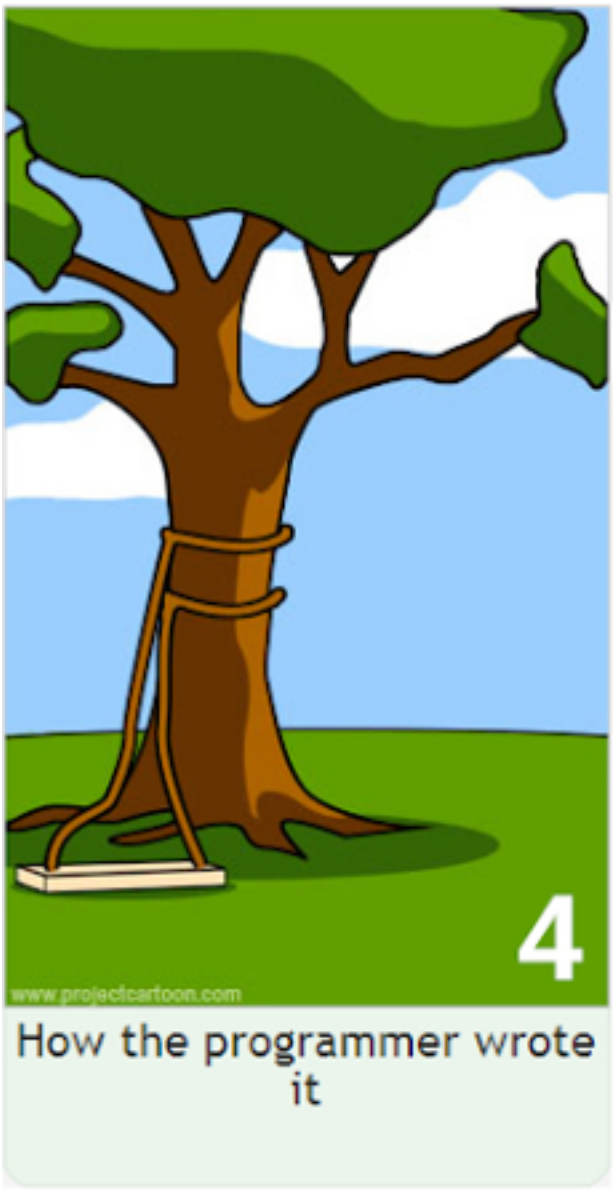
0.14 1-How the customer explained it

0.15 2-How the project leader understood it.



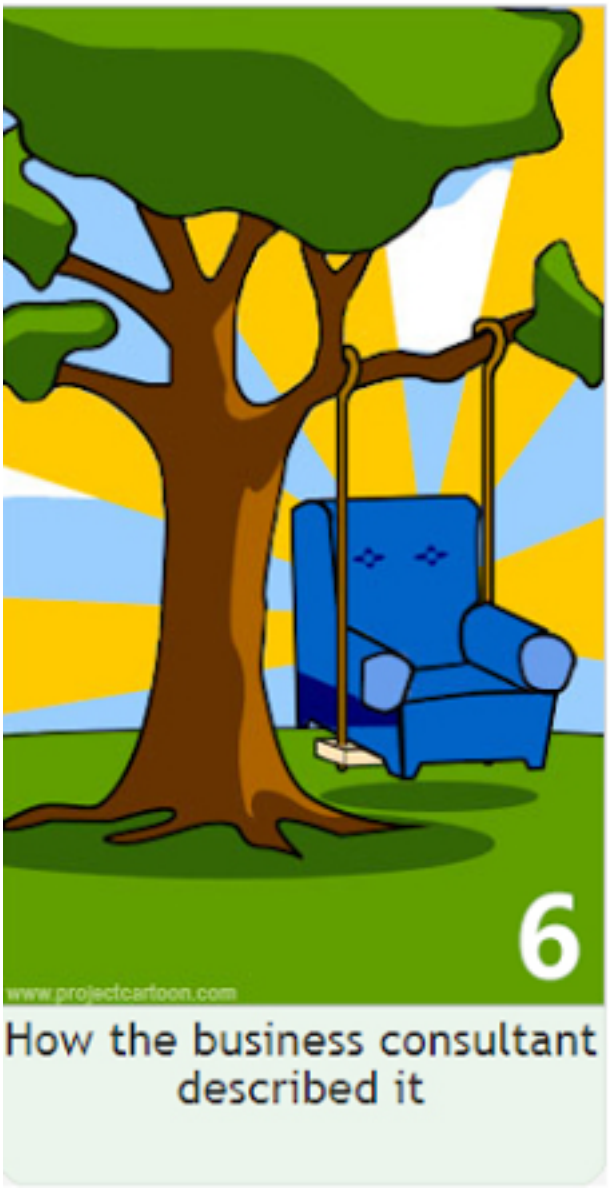
0.16 3- How the analyst designed it

0.17 4-How the programmer wrote it

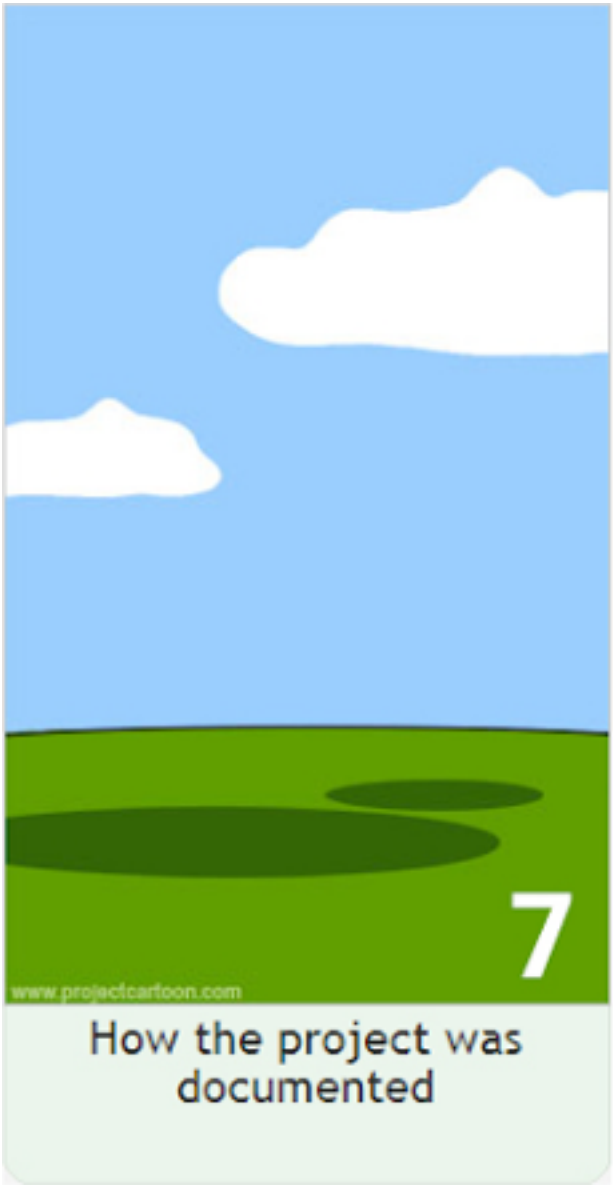


0.18 5-What the beta testers received

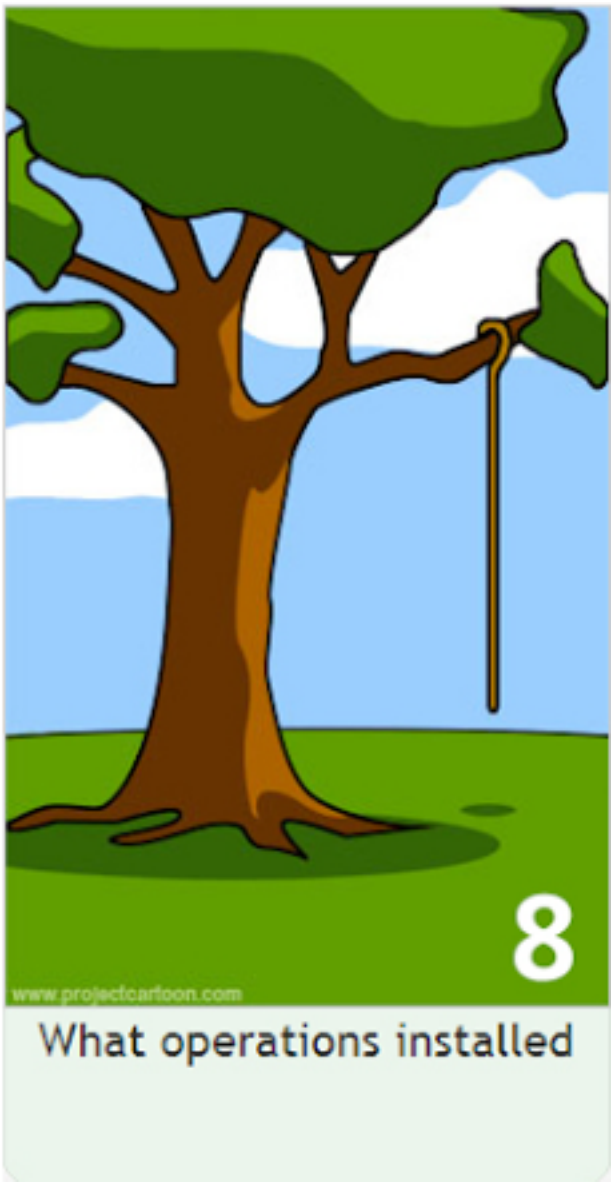
0.19 6-How the business consultant described it



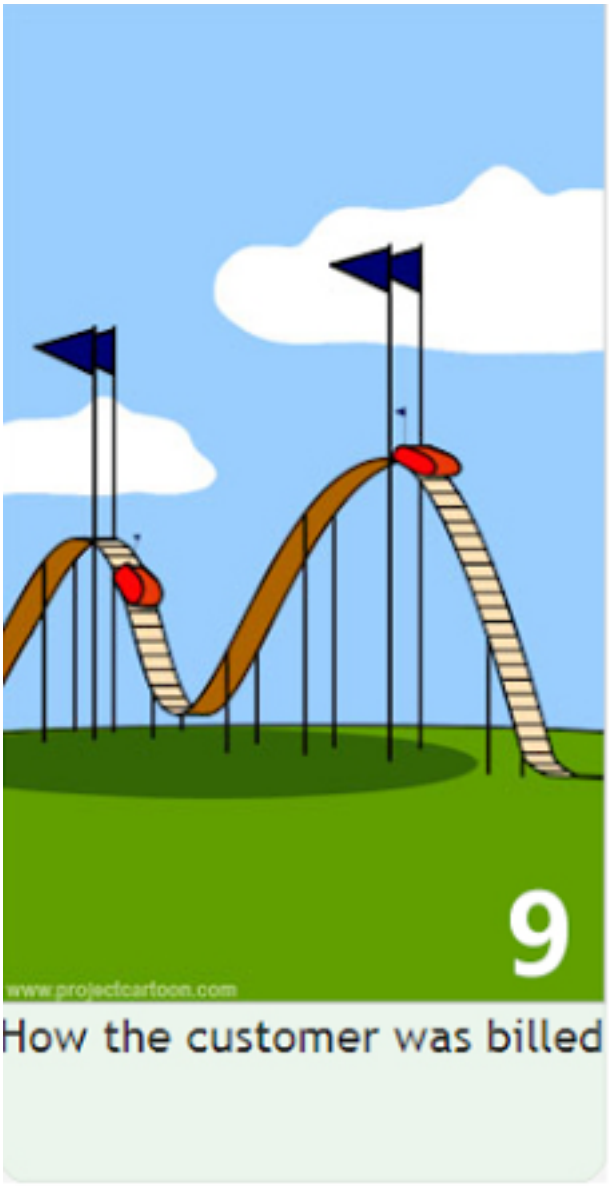
0.20 7- How the project was documented



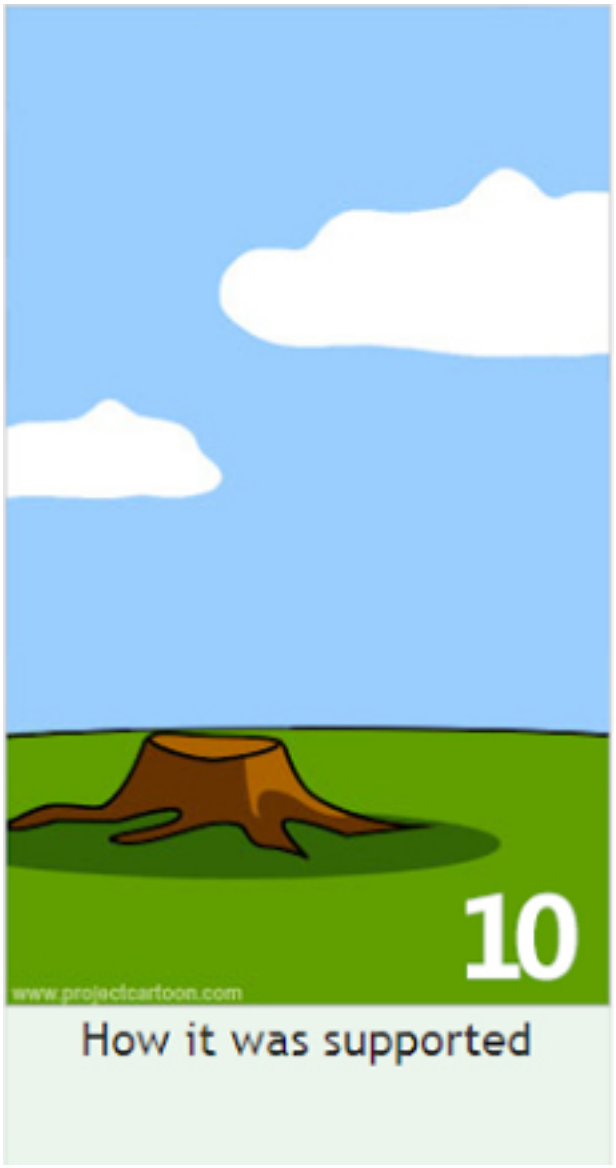
0.21 8-What operations installed



0.22 9-How the customer was billed



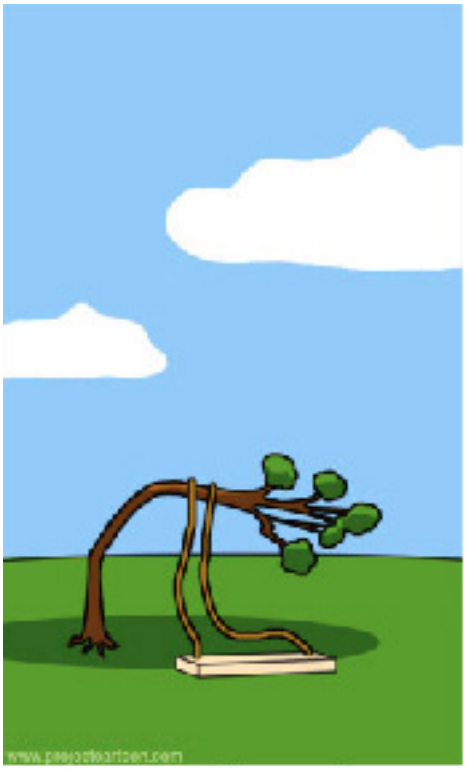
0.23 10-How it was supported



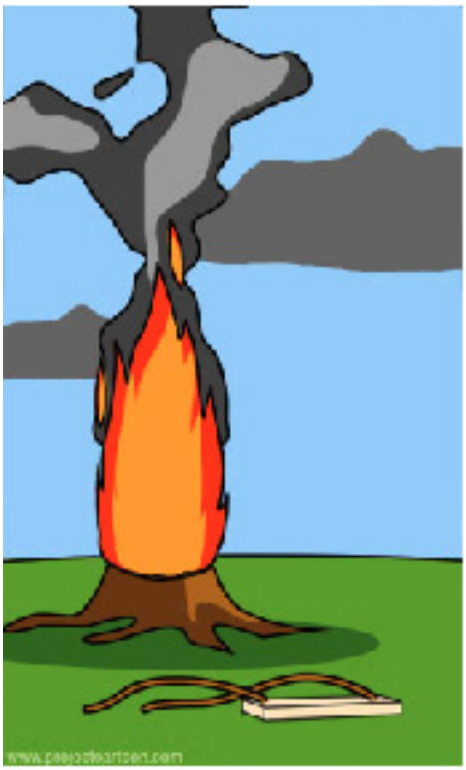
0.24 11-What marketing advertised

0.25 12-What the customer really needed

0.26 13-The disaster recover plan



0.27 14- What the digg effect can do to your site



0.28 15- (Finally) When it was delivered

0.29 Need for a good programming method

- **Common problems**
  - Why does it take so long?
  - Why are development costs so high?
  - Why can't find all faults before delivery?
  - Why can't we measure development?

NIST reported that even though 50 percent of software development budgets go to testing, flaws in software still cost the U.S. economy $59.5 billion annually.*

Updated NIST Software Uses Combination Testing to Catch Bugs Fast and Easy | NIST[4]

0.30 What is Software Engineering? (1)

- The process of **solving customers' problems** by the
  - systematic development and evolution of large,
  - high-quality software systems within

[4] https://www.nist.gov/news-events/news/2010/11/updated-nist-software-uses-combination-testing-catch-bugs-fast-and-easy

    * cost,
    * time and
    * other constraints

0.31 What is Software Engineering? (2)

- **Solving customers' problems**
  - The goal
    * Sometimes the solution is to **buy**, not build
    * Adding unnecessary features often makes software worse
    * Software engineers must communicate effectively to identify and understand the problem

---

0.32 What is Software Engineering? (3)

- **Systematic development and evolution**
  - An engineering process involves applying *well understood techniques* in a organized and *disciplined* way
  - Many well-accepted practices have been formally standardized
    * e.g. by the IEEE or ISO
  - Most development work is evolution

0.33 What is Software Engineering? (4)

- **Large, high quality software systems**
  - Software engineering techniques are needed because large systems *cannot be completely understood* by one person
  - Teamwork and co-ordination are required
  - *Key challenge:* Dividing up the work and ensuring that the parts of the system work properly together
  - The end-product must be of sufficient quality

0.34 What is Software Engineering? (5)

- **Cost, time and other constraints**
  - Finite resources
  - The benefit must outweigh the cost
  - Others are competing to do the job cheaper and faster
  - Inaccurate estimates of cost and time have caused many project failures

---

0.35 The Software Engineering Profession

- **The term Software Engineering was coined in 1968**
  - People began to realize that the principles of engineering should be applied to software development
- **Engineering is a licensed profession**
  - In order to protect the public
  - Engineers design artifacts following well accepted practices which involve the application of science, mathematics and economics
  - Ethical practice is also a key tenet of the profession

- **In many countries, much software engineering does not require an engineering licence, but is still engineering**

0.36 Software Engineering Code of Ethics

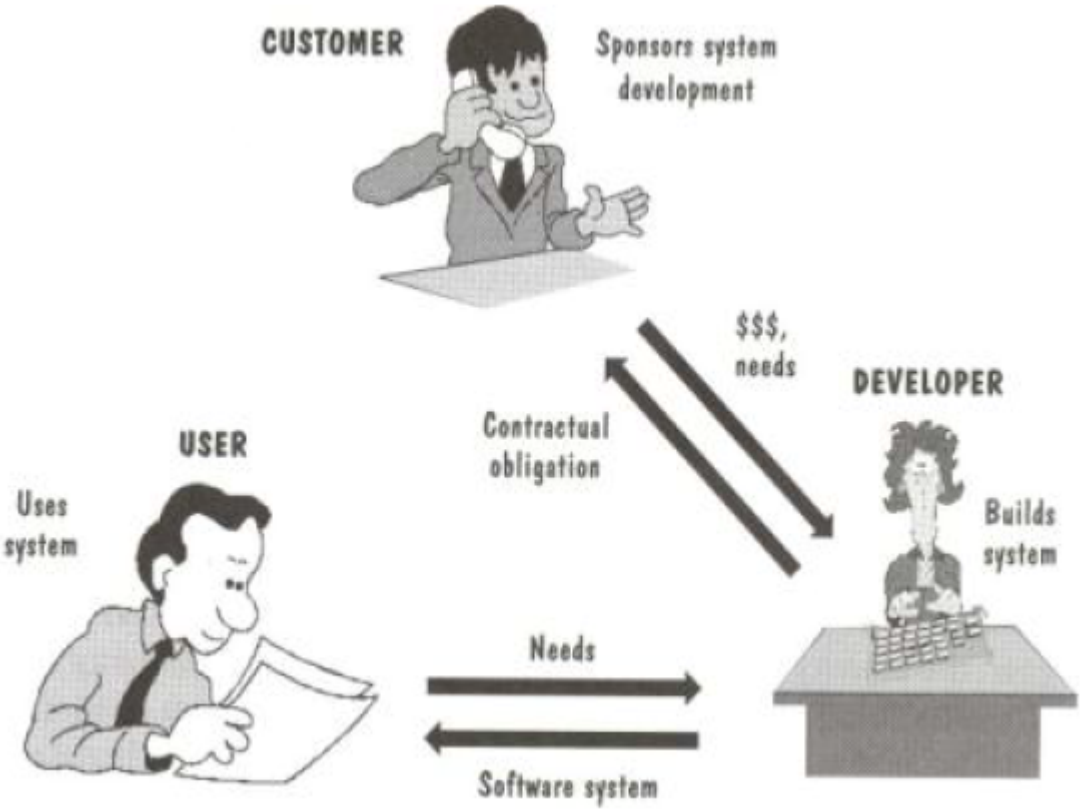
- ***Software engineers shall***
  - Act consistently with **public interest**
  - Act in the **best interests of their clients**
  - Develop and maintain with the **highest standards** possible
  - Maintain **integrity and independence**
  - Promote an **ethical** approach in **management**
  - Advance the integrity and **reputation of the profession**
  - Be fair and **supportive to colleagues**
  - Participate in **lifelong learning**

0.37 Software Quality

- **Usability**
  - Users can learn it and fast and get their job done easily
- **Efficiency**
  - It doesn't waste resources such as CPU time and memory
- **Reliability**
  - It does what it is required to do without failing
- **Maintainability**
  - It can be easily changed
- **Reusability**
  - Its parts can be used in other projects, so reprogramming is not needed

---

0.38 Software Quality and Stakeholders



0.39 Software Quality and Stakeholders

- **Customer (those who pay):**
  - solves problems at an acceptable cost in terms of money paid and resources used
- **User**
  - easy to learn; efficient to use; helps get work done
- **Developer**
  - easy to design; easy to maintain; easy to reuse its parts
- **Development manager**
  - sells more and pleases customers while costing less to develop and maintain

---

0.40 Software Quality Metrics – User

- A program must do its **job correctly**. It must be useful and - usable
- A program must **run** as **fast** as necessary (Real-time constraints)
- A program must not **waste system resources**(processor time, - memory, disk capacity, network capacity) too much
- It must be **reliable**
- It must be easily **updated**
- A good software must have sufficient documentation (users manual)

0.41 Software Quality Metrics – SW Developer

- Source code must be **readable and understandable**

- It must be **easy** to **maintain** and **update** the program
- A program must consist of **independent modules**
- An **error** may not affect other parts of a program (Locality of errors)
- Modules of the program must be **reusable** in other projects
- A software project must meet its **deadline**
- Good software must have **sufficient documentation**

0.42 Software Quality Metrics - OOP

- OOP techniques ensure high-quality programs
- While designing and coding a program, these quality metrics must always be considered

---

0.43 Software Quality: Conflicts and Objectives

- **The different qualities can conflict**
  - Increasing efficiency can reduce maintainability or reusability
  - Increasing usability can reduce efficiency
- **Setting objectives for quality is a key engineering activity**
  - You then design to meet the objectives
  - Avoids "over-engineering" which wastes money

0.44 Software Engineering Projects

- **Most projects are evolutionary or maintenance projects, involving work on legacy systems**
  - *Corrective* projects: fixing defects
  - *Adaptive* projects: changing the system in response to changes in
    - Operating system
    - Database
    - Rules and regulations
  - *Enhancement* projects: adding new features for users
  - *Reengineering* or *perfective* projects: changing the system internally so it is more maintainable

---

0.45 Software Engineering Projects Start Points

Green Field Development	Brownfield Development
Start afresh	Build on existing code
Choose your technology	Technology already chosen
Use your best ideas, patterns, techniques	Understand previous developers' code
Learn from mistakes	Live with mistakes

0.46 Software Engineering Projects Start Points

	Requirements must be determined	Clients have produced rec
New development, Green Field Project	A	B
Evolution of Existing System, Brown Field Project	C	D

0.47 Activities Common to Software Projects

- Requirements and specification
  - Includes
    - Domain analysis
    - Defining the problem
    - Requirements gathering
      - Obtaining input from as many sources as possible
    - Requirements analysis
      - Organizing the information
    - Requirements specification
      - Writing detailed instructions about how the software should behave

---

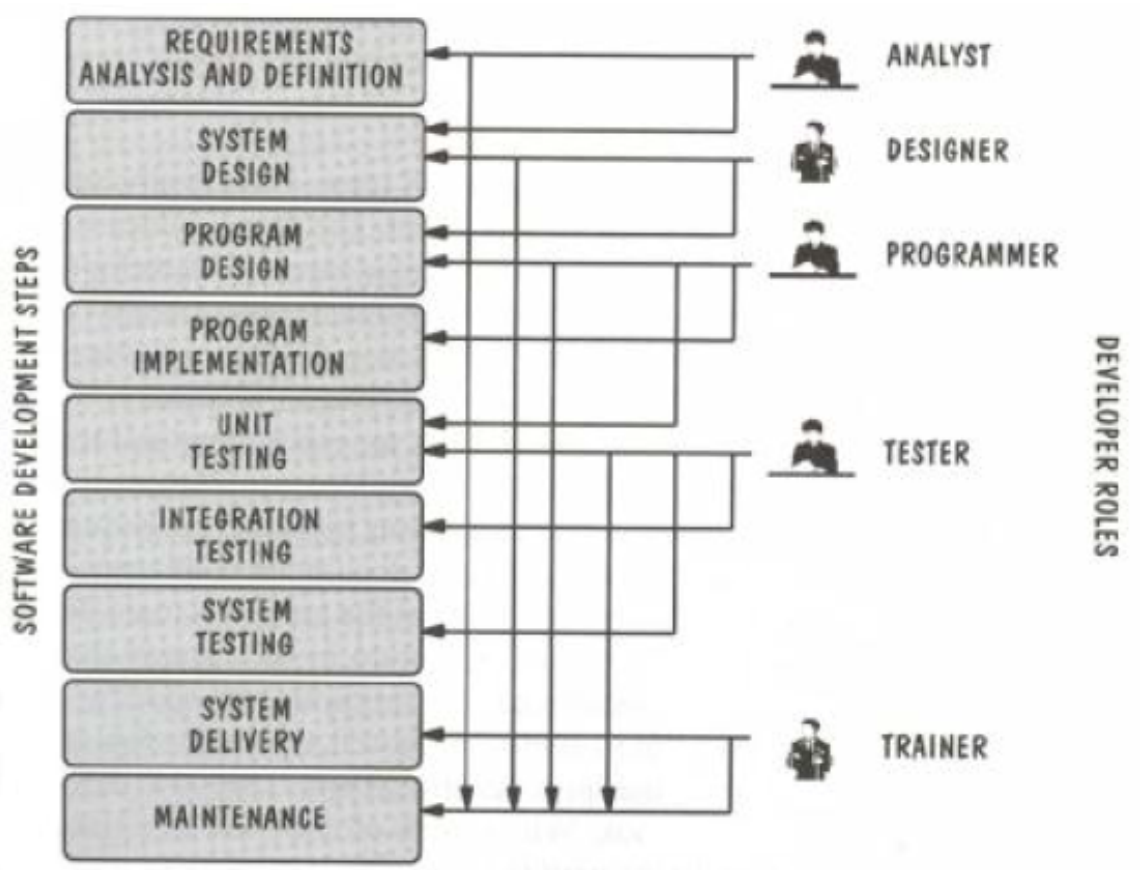
0.48 Activities Common to Software Projects

- **Design**
  - Deciding how the requirements should be implemented, using the available technology
  - Includes:
    - Systems engineering: Deciding what should be in hardware and what in software
    - Software architecture: Dividing the system into subsystems and deciding how the subsystems will interact
    - Detailed design of the internals of a subsystem
    - User interface design
    - Design of databases

0.49 Activities Common to Software Projects

- **Modeling**
  - Creating representations of the domain or the software
    - Use case modeling
    - Structural modeling
    - Dynamic and behavioural modeling
- **Programming**
- **Quality assurance**
  - Reviews and inspections
  - Testing
- **Deployment**
- **Managing the process**

0.50 Software Projects Development Team



0.51 Software Life Cycle

- Requirements Phase
- Specification Phase
- Design Phase
- Implementation Phase
- Integration Phase
- Maintenance Phase
- Retirement Phase

0.52 Requirements Phase

- Defining constraints
  - Functions
  - Due dates
  - Costs
  - Reliability
  - Size
- Types
  - Functional
  - Non-Functional

0.53 Specification Phase

- Documentation of requirements
  - Inputs & Outputs
  - Formal
  - Understandable for user & developer
  - Usually functional requirements (what to do)
  - Base for testing & maintenance
- The contract between customer & developer

0.54 Design Phase

- Defining Internal structure (how to do)
- Has some levels (or types of docs)
  - Architectural design
  - Detailed design
- Important
  - To backtrack the aims of decisions
  - To easily maintain

0.55 Implementation and Integration Phases

- Implementation phase: Simply coding
  - Unit tests
    - For verification
- Combining modules
- System tests
  - For validation
- Quality tests

0.56 Maintenance Phase

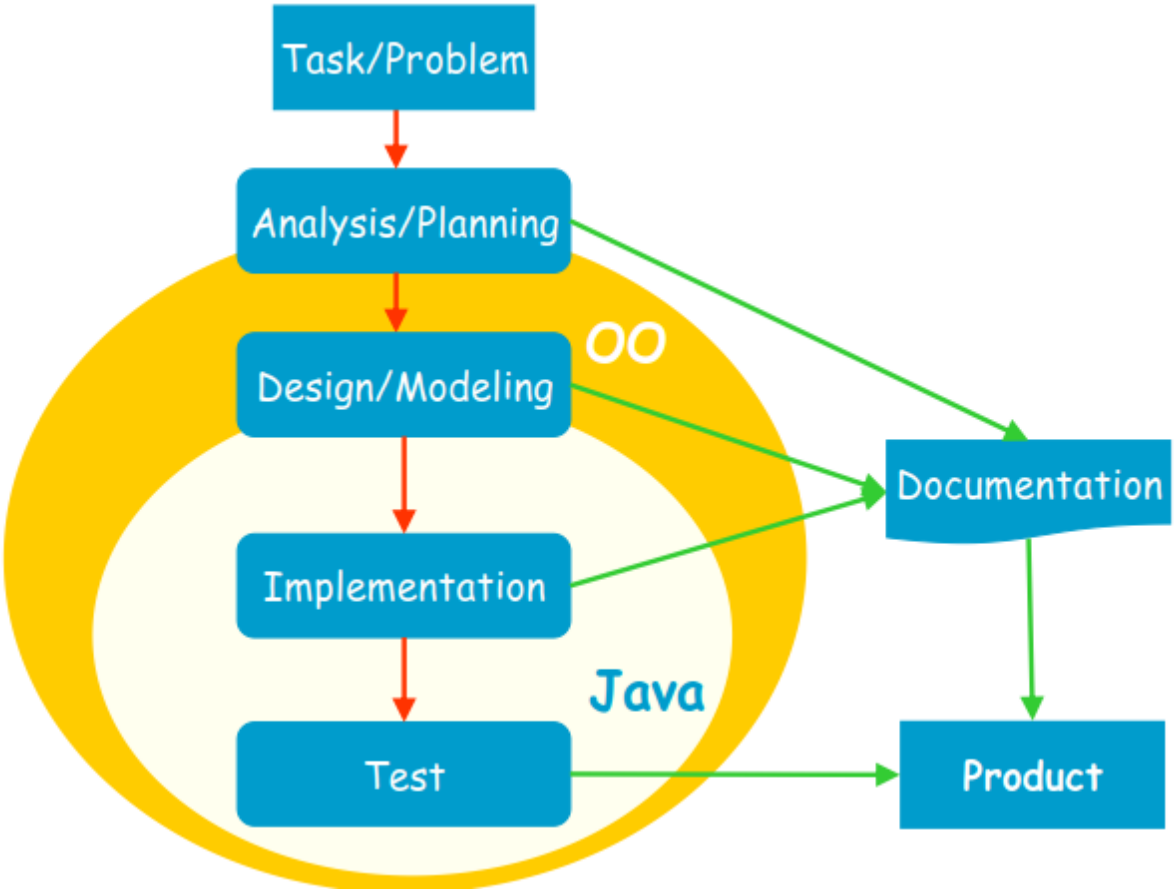
- Corrective
- Enhancement
  - Perfective
  - Adaptive
- Usually maintainers are not the same people with developers.
- The only input is (in general) the source code of the software

---

0.57 Retirement Phase

- When the cost of maintenance is not effective.
  - Changes are so drastic, that the software should be redesigned.
- So many changes may have been made.
- The update frequency of docs is not enough.
- The hardware (or OS) will be changed.

0.58 Software Development Process



0.59 Software Development Process

- **Analysis**: Understanding requirements. They may change during (or after) development of the system! Building the programming team.
- **Design**: Identifying the key concepts involved in a solution and creation of the models.
  - This stage has a strong effect on the quality of the software. Therefore, before the coding, verification of the created model must be done.
  - Design process is connected with the programming scheme. Here, our design style is object-oriented.

0.60 Software Development Process

- **Coding**: The solution (model) is expressed in a program. In this course we will use Java.
- Documentation: Each phase of a software project must be clearly explained. A users manual should also be written.
- **Test**: the behavior of the program for possible inputs must be examined.

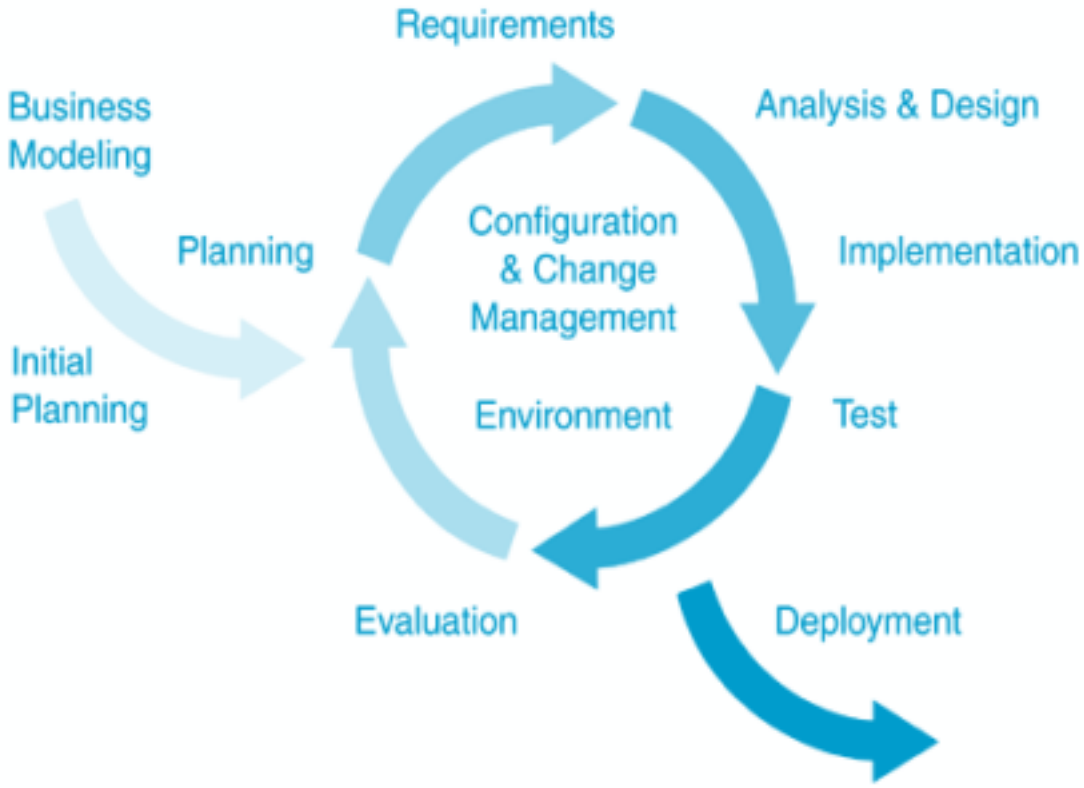
These steps are important design principles and design patterns, which help us developing high-quality software. The Unified Modeling Language (UML) is useful to express the model.

0.61 Unified Process (UP)

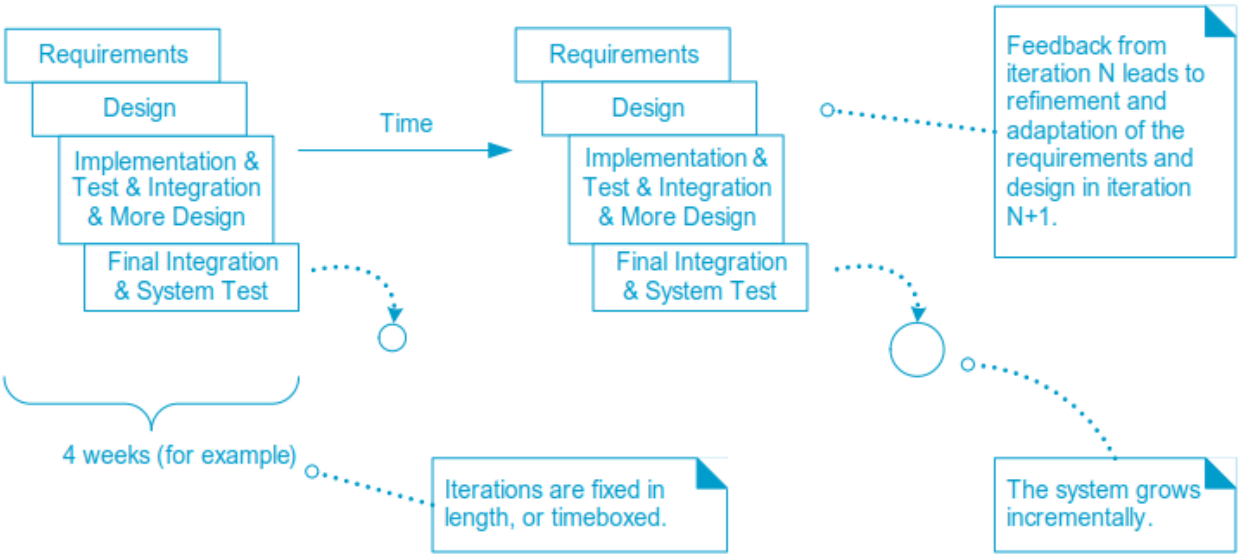
- The UP promotes several best practices.

- Iterative
- Incremental
- Risk-driven

0.62 Unified Process (UP)



0.63 Unified Process (UP)



0.64 Object Orientation

0.65 OOP Concepts in Java

OOP stands for Object-Oriented Programming. OOP is a programming paradigm in which every program is follows the concept of object. In other words, OOP is a way of writing programs based on the object concept.

The object-oriented programming paradigm has the following core concepts.

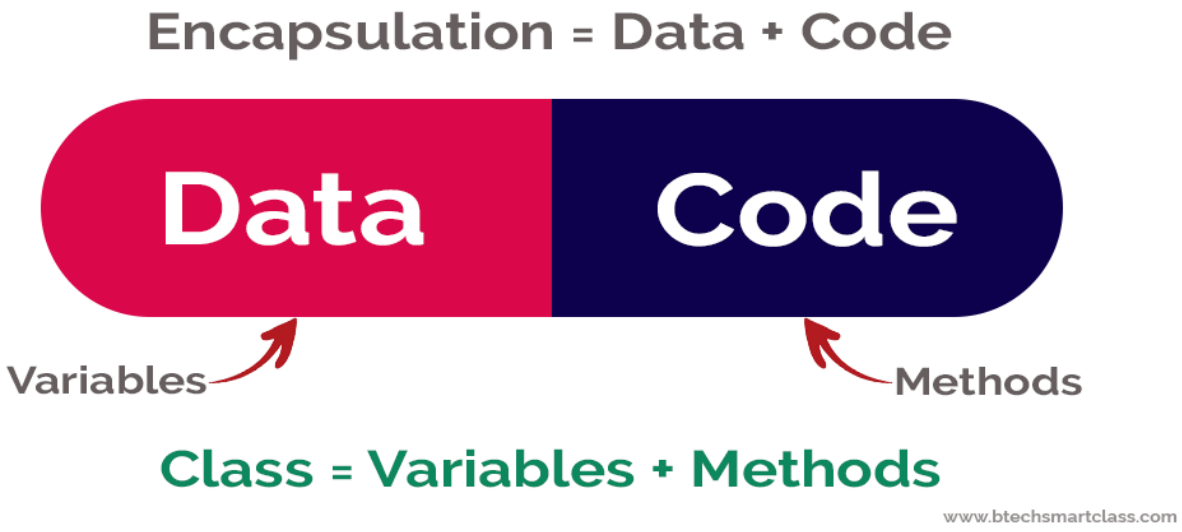
- Encapsulation
- Inheritance
- Polymorphism
- Abstraction

The popular object-oriented programming languages are Smalltalk, C++, Java, PHP, C#, Python, etc.

0.66 Encapsulation

Encapsulation is the process of combining data and code into a single unit (object / class). In OOP, every object is associated with its data and code. In programming, data is defined as variables and code is defined as methods. The java programming language uses the class concept to implement encapsulation.

0.67 Encapsulation



0.68 Inheritance

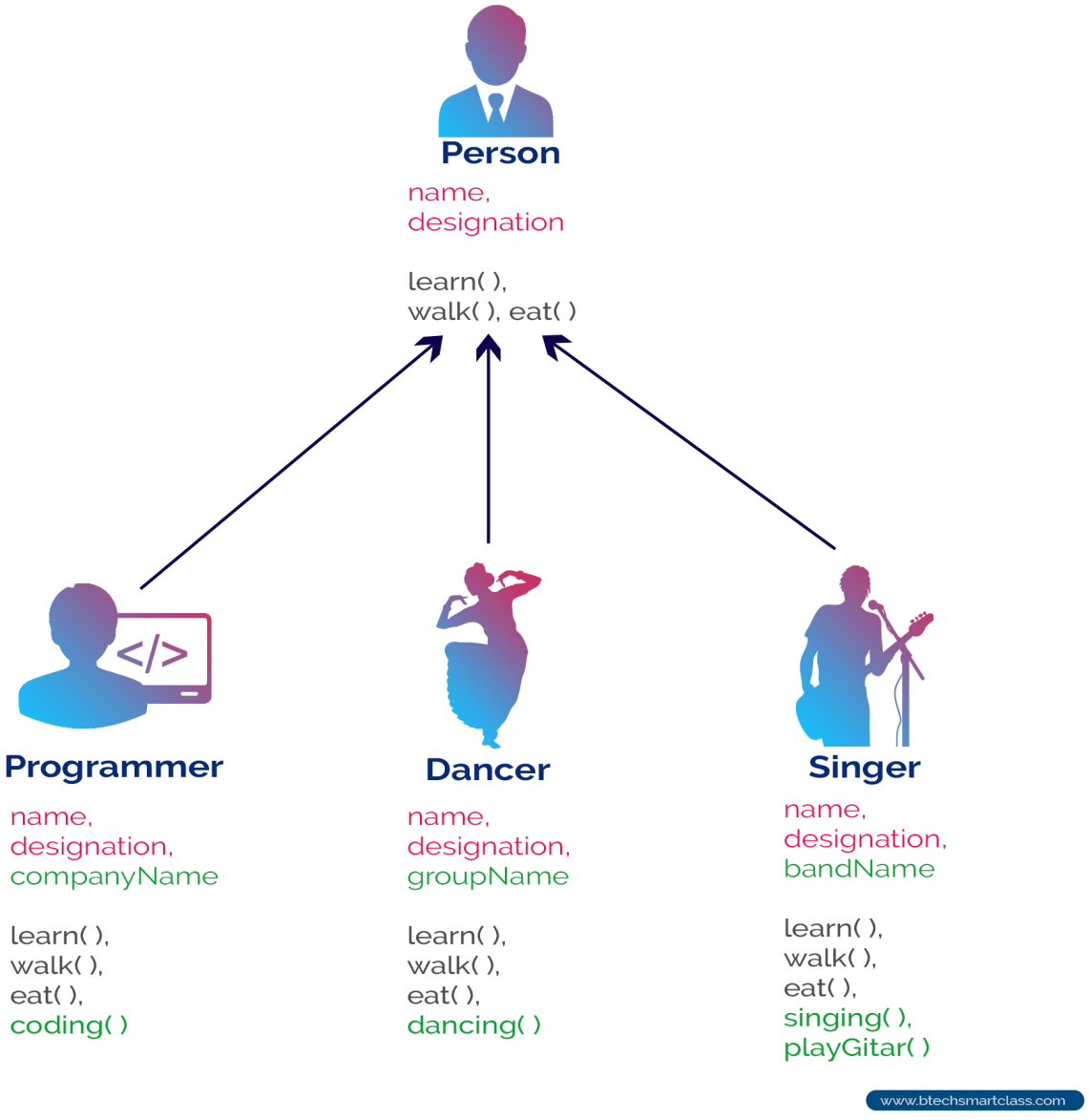
Inheritance is the process of acquiring properties and behaviors from one object to another object or one class to another class. In inheritance, we derive a new class from the existing class. Here, the new class

acquires the properties and behaviors from the existing class. In the inheritance concept, the class which provides properties is called as parent class and the class which recieves the properties is called as child class. The parent class is also known as base class or supre class. The child class is also known as derived class or sub class.

In the inheritance, the properties and behaviors of base class extended to its derived class, but the base class never receive properties or behaviors from its derived class.

In java programming language the keyword extends is used to implement inheritance.

0.69 Inheritance



0.70 Polymorphism

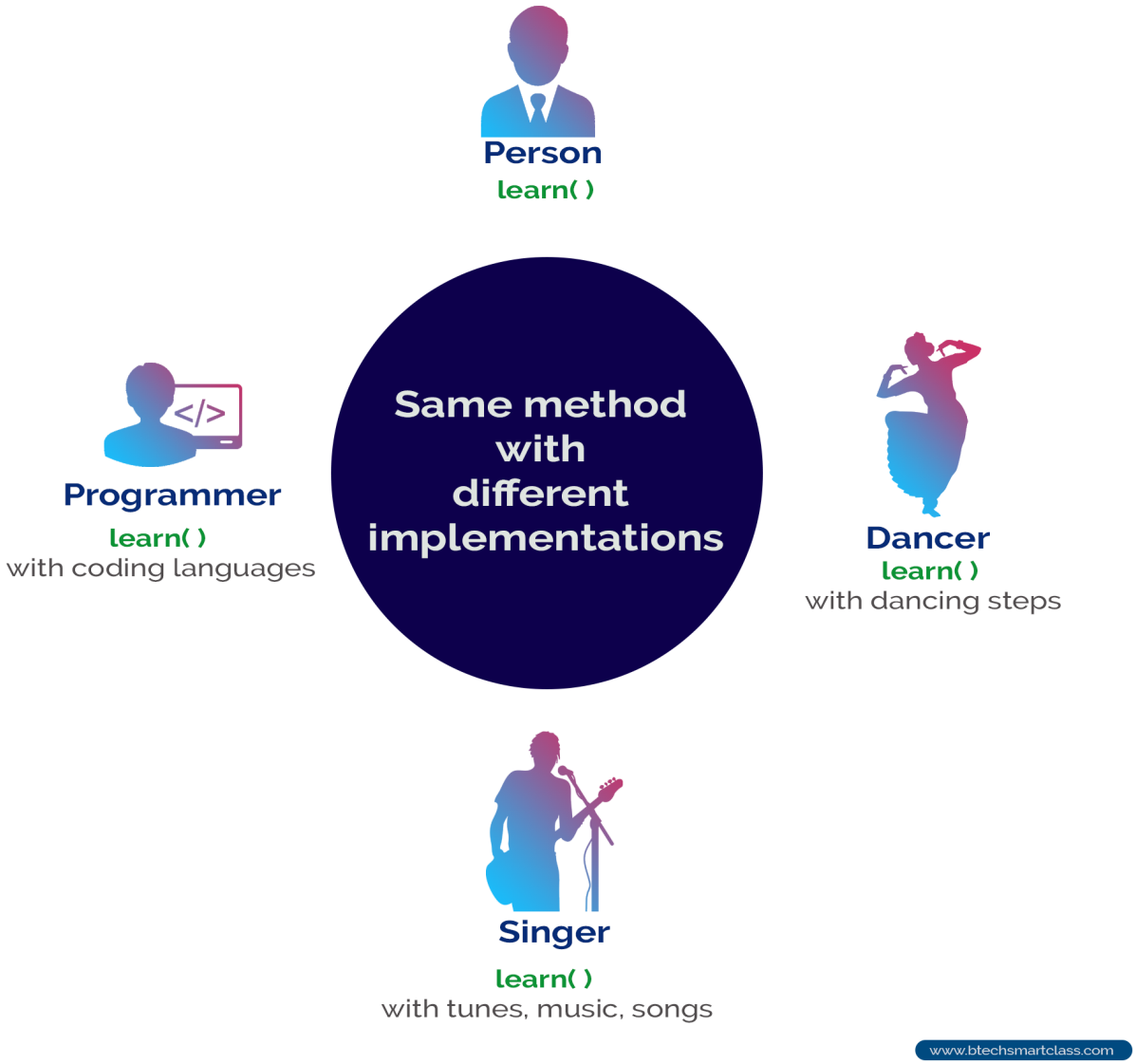
Polymorphism is the process of defining same method with different implementation. That means creating multiple methods with different behaviors.

The java uses method overloading and method overriding to implement polymorphism.

Method overloading - multiple methods with same name but different parameters.

Method overriding - multiple methods with same name and same parameters.

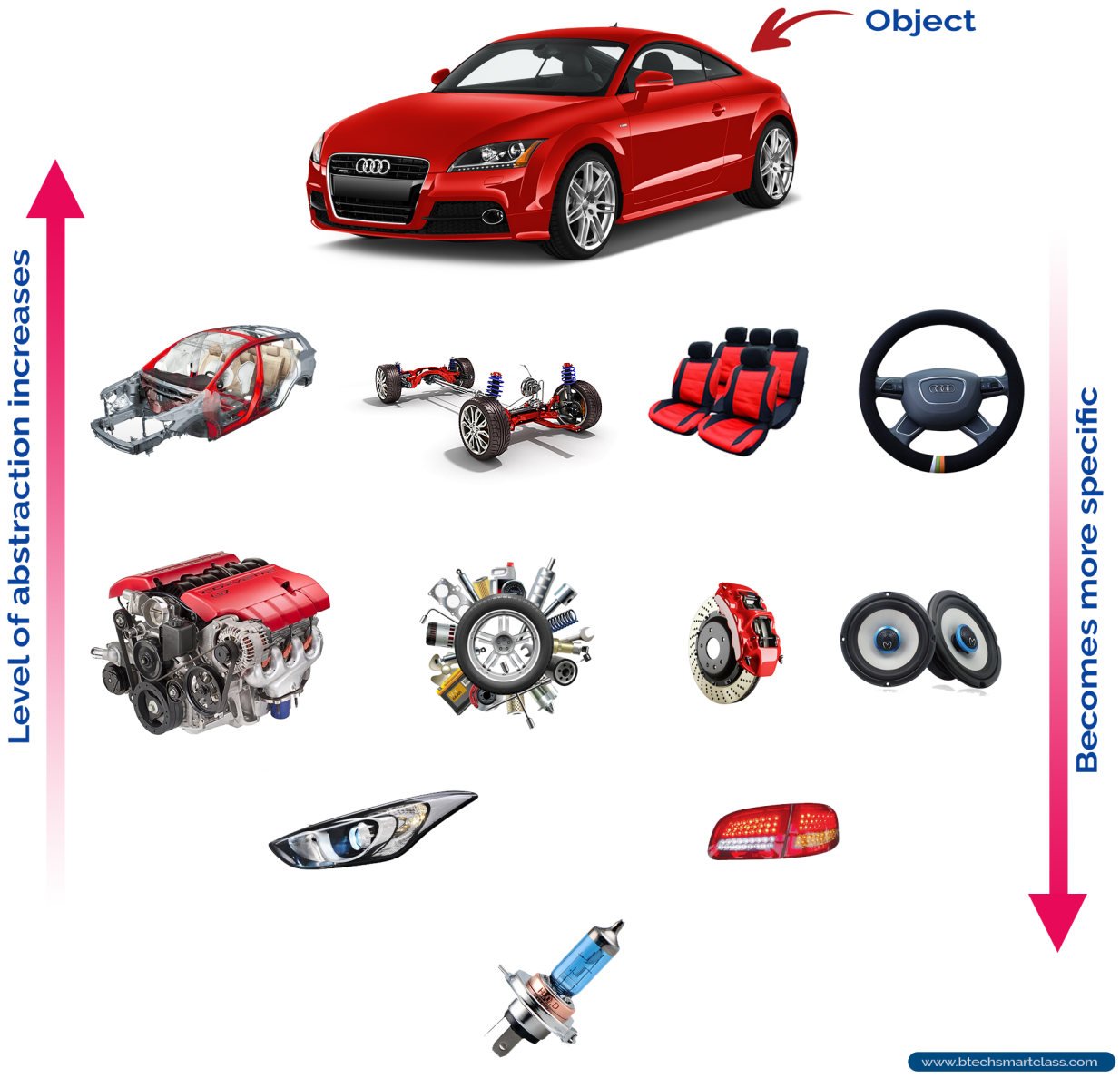
0.71 Polymorphism



0.72 Abstraction

Abstraction is hiding the internal details and showing only esential functionality. In the abstraction concept, we do not show the actual implemention to the end user, instead we provide only esential things. For example, if we want to drive a car, we does not need to know about the internal functionality like how wheel system works? how brake system works? how music system works? etc.

0.73 Abstraction



0.74 Why Object Technology

- Expectations are,
- Reducing the effort, complexity, and cost of development and maintenance of software systems.

- Reducing the time to adapt an existing system (quicker reaction to changes in the business environment): Flexibility, reusability.
- Increasing the reliability of the system.

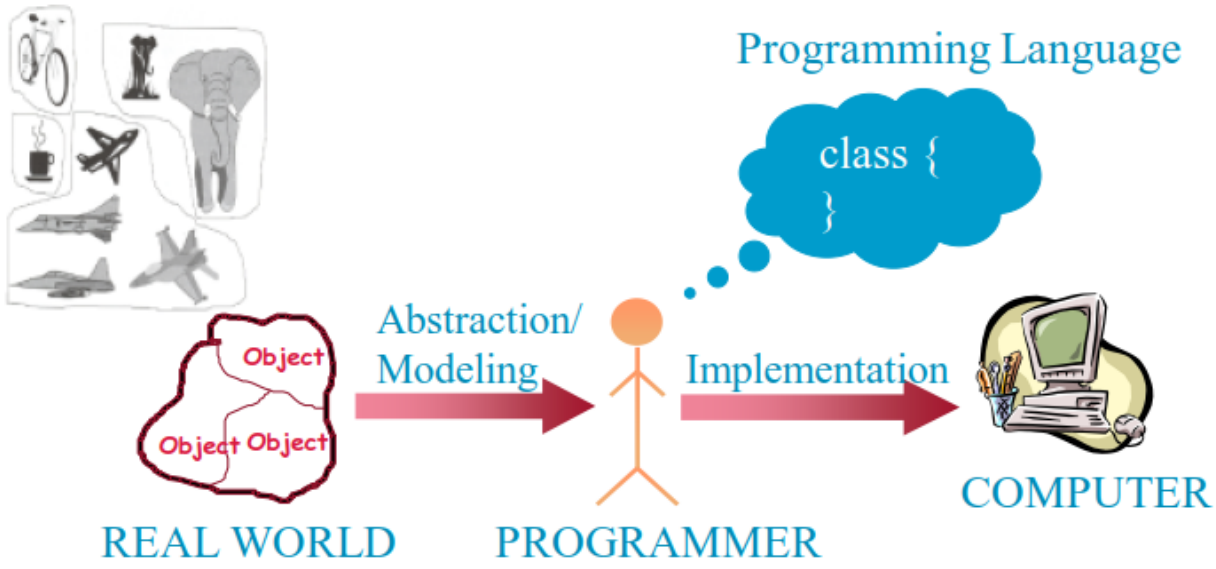
0.75 What is Programming?

- A programming language provides a way to express concepts.

- Program development involves creating models of real world - situations and building computer programs based on these models.
- Computer programs describe the method of implementing the model.
- Computer programs may contain computer world representations of the things that constitute the solutions of real world problems.

0.76 What is Programming?

If successful, this medium of expression (the object-oriented way) will be significantly easier, more flexible, and efficient than the alternatives as problems grow larger and more complex



0.77 Why JAVA

- Java supports writing high quality programs (pure OO)
- Provides an easy-to-use language
- Provides an interpreted environment for
  - Improved development speed
  - Code portability
- Simple
- Architecture Neutral and Portable
- Robust and Secure
- High Performance

0.78 Why JAVA

- Write less code
  - can be four times smaller than the same program written in C++
- Write better code
  - encourages good coding practices, garbage collection for avoiding memory leaks, wide-ranging, easily extendible API
- Avoid platform dependencies
- Write once, run anywhere
- Gained popularity in gadgets such as
  - PDAs, cell phones etc.

0.79 Learning JAVA

- Many syntax and grammar rules
- Learning how to write "good programs"
- Focusing on concepts and not get lost in language-technical - details
- Paying attention to design techniques rather than details
- Building an effective programming scheme
- Practicing, practicing and practicing!
- Consequently, new and better ways of building systems

0.80 Java Buzz Words

- Simple
- Secure
- Portable
- Object-oriented
- Robust
- Architecture-neutral (or) Platform Independent
- Multi-threaded
- Interpreted
- High performance
- Distributed
- Dynamic

0.81 Simple

Java programming language is very simple and easy to learn, understand, and code. Most of the syntaxes in java follow basic programming language C and object-oriented programming concepts are similar to C++. In a java programming language, many complicated features like pointers, operator overloading, structures, unions, etc. have been removed. One of the most useful features is the garbage collector it makes java more simple.

0.82 Secure

Java is said to be more secure programming language because it does not have pointers concept, java provides a feature "applet" which can be embedded into a web application. The applet in java does not allow access to other parts of the computer, which keeps away from harmful programs like viruses and unauthorized access.

0.83 Portable

Portability is one of the core features of java which enables the java programs to run on any computer or operating system. For example, an applet developed using java runs on a wide variety of CPUs, operating systems, and browsers connected to the Internet.

0.84 Object-oriented

Java is said to be a pure object-oriented programming language. In java, everything is an object. It supports all the features of the object-oriented programming paradigm. The primitive data types java also implemented as objects using wrapper classes, but still, it allows primitive data types to archive high-performance.

0.85 Robust

Java is more robust because the java code can be executed on a variety of environments, java has a strong memory management mechanism (garbage collector), java is a strictly typed language, it has a strong set of exception handling mechanism, and many more.

0.86 Architecture-neutral (or) Platform Independent

Java has invented to archive "write once; run anywhere, any time, forever". The java provides JVM (Java Virtual Machine) to to archive architectural-neutral or platform-independent. The JVM allows the java program created using one operating system can be executed on any other operating system.

0.87 Multi-threaded

Java supports multi-threading programming, which allows us to write programs that do multiple operations simultaneously.

0.88 Interpreted

Java enables the creation of cross-platform programs by compiling into an intermediate representation called Java bytecode. The byte code is interpreted to any machine code so that it runs on the native machine.

0.89 High performance

Java provides high performance with the help of features like JVM, interpretation, and its simplicity.

0.90 Distributed

Java programming language supports TCP/IP protocols which enable the java to support the distributed environment of the Internet. Java also supports Remote Method Invocation (RMI), this feature enables a program to invoke methods across a network.

0.91 Dynamic

Java is said to be dynamic because the java byte code may be dynamically updated on a running system and it has a dynamic memory allocation and deallocation (objects and garbage collector).

0.92 The Basics of Java

History

- The first object oriented programming language was Simula-67
  - designed to allow programmers to write simulation programs
- In the early 1980's, Smalltalk was developed at Xerox PARC
  - New syntax, large open-source library of reusable code, bytecode, platform independence, garbage collection.

0.93 The Basics of Java

- late 1980's, C++ was developed by B. Stroustrup,
  - Recognized the advantages of OO but also recognized that there were tremendous numbers of C programmers
- In 1991, engineers at Sun Microsystems started a project to design a language that could be used in consumer 'smart devices': Oak
  - When the Internet gained popularity, Sun saw an opportunity to exploit the technology.
  - The new language, renamed Java, was formally presented in 1995 at the SunWorld '95 conference.

0.94 Java documentation

- Looking up classes and methods is an essential skill
  - Looking up unknown classes and methods will get you a long way towards understanding code
- Java documentation can be automatically generated by a program called Javadoc
  - Documentation is generated from the code and its comments
  - You should format your comments as shown in some of the book's examples
    - These may include embeded html

0.95 Characters and Strings

- Character is a class representing Unicode characters
  - More than a byte each
  - Represent any world language
- char is a primitive data type containing a Unicode character
- `String` is a class containing collections of characters
- + is the operator used to concatenate strings

0.96 Arrays and Collections

- Arrays are of fixed size and lack methods to manipulate them
- ArrayList is the most widely used class to hold a collection of other objects
  - More powerful than arrays, but less efficient
- Iterators are used to access members of Vectors
  - Enumerations were formally used, but were more complex

```
a = new ArrayList();
Iterator i = a.iterator();
while(i.hasNext())
{
   aMethod(i.next());
}
```

0.97 Casting

- Java is very strict about types
  - If variable v is declared to have type X, you can only invoke operations on v that are defined in X or its *superclasses*
    - Even though an instance of a subclass of X may be actually stored in the variable
  - If you know an instance of a subclass is stored, then you can cast the variable to the subclass
    - E.g. if I know a Vector contains instances of String, I can get the next element of its Iterator using: `(String)i.next();`
    - To avoid casting you could also have used templates:: `a = ArrayList<String>; i=a.iterator(); i.next()`

0.98 Exceptions

- **Anything that can go wrong should result in the raising of an Exception**
  - Exception is a class with many subclasses for specific things that can go wrong
- **Use a try - catch block to trap an exception**

```
try
{
   // some code
}
catch (ArithmeticException e)
{
  // code to handle division by zero
}
```

0.99 Interfaces

- **Like abstract classes, but cannot have executable statements**
  - Define a set of operations that make sense in several classes
  - Abstract Data Types
- **A class can implement any number of interfaces**
  - It must have concrete methods for the operations
- **You can declare the type of a variable to be an interface**
  - This is just like declaring the type to be an abstract class
- **Important interfaces in Java's library include**
  - Runnable, Collection, Iterator, Comparable, Cloneable

0.100 Packages and importing

- **A package combines related classes into subsystems**
  - All the classes in a particular directory
- **Classes in different packages can have the same name**
  - Although not recommended

- **Importing a package is done as follows:**
  - `import finance.banking.accounts.*;`

0.101 Access control

- **Applies to methods and variables**
  - *public*
    - Any class can access
  - *protected*
    - Only code in the package, or subclasses can access
  - *(blank)*
    - Only code in the package can access
  - *private*
    - Only code written in the class can access
    - Inheritance still occurs!

0.102 Threads and concurrency

- **Thread**:
  - Sequence of executing statements that can be running concurrently with other threads
- **To create a thread in Java**:
  - Create a class implementing Runnable or extending Thread
  - Implement the run method as a loop that does something for a period of time
  - Create an instance of this class
  - Invoke the start operation, which calls run

0.103 Programming Style Guidelines

- **Remember that programs are for people to read**
  - Always choose the simpler alternative
  - Reject clever code that is hard to understand
  - Shorter code is not necessarily better
- **Choose good names**
  - Make them highly descriptive
  - Do not worry about using long names

0.104 Programming style

- **Comment extensively**
  - Comment whatever is non-obvious
  - Do not comment the obvious
  - Comments should be 25-50% of the code
- **Organize class elements consistently**
  - Variables, constructors, public methods then private methods
- **Be consistent regarding layout of code**

0.105 Programming style

- Avoid duplication of code
  - Do not “clone” if possible

    * Create a new method and call it
    * Cloning results in two copies that may both have bugs
        · When one copy of the bug is fixed, the other may be forgotten

---

0.106 Programming style

- Adhere to good object oriented principles
  - E.g. the 'isa rule'
- Prefer **private** as opposed to **public**
- Do not mix user interface code with non-user interface code
  - Interact with the user in separate classes
    * This makes non-UI classes more reusable

0.107 Difficulties and Risks in Programming

- **Language evolution and deprecated features**:
  - Java is evolving, so some features are 'deprecated' at every release
- **Efficiency can be a concern in some object oriented systems**
  - Java can be less efficient than other languages
    * VM-based
    * Dynamic binding

0.108 C++ vs Java

Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in Windows-based, web-based, enterprise, and mobile applications.

0.109 C++ vs Java

Comparison Index	C++	Java
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by using interfaces in java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.

0.110 C++ vs Java

Comparison Index	C++	Java
Goto	C++ supports the goto statement.	Java doesn't support the goto statement.
Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.	Java uses both compiler and interpreter. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is platform-independent.

0.111 C++ vs Java

Comparison Index	C++	Java
Pointers	C++ supports pointers. You can write a pointer program in C++.	Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in java.
Design Goal	C++ was designed for systems and applications programming. It was an extension of the C programming language.	Java was designed and created as an interpreter for printing systems but later extended as a support network computing. It was designed to be easy to use and accessible to a broader audience.

0.112 C++ vs Java

Comparison Index	C++	Java
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.

0.113 C++ vs Java

Comparison Index	C++	Java
Documentation comment	C++ doesn't support documentation comments.	Java supports documentation comment (/** … */) to create documentation for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not to override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.

0.114 C++ vs Java

Comparison Index	C++	Java
unsigned right shift »>	C++ doesn't support »> operator.	Java supports unsigned right shift »> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like » operator.
Inheritance Tree	C++ always creates a new inheritance tree.	Java always uses a single inheritance tree because all classes are the child of the Object class in Java. The Object class is the root of the inheritance tree in java.

0.115 C++ vs Java

Comparison Index	C++	Java
Hardware	C++ is nearer to hardware.	Java is not so interactive with hardware.
Object-oriented	C++ is an object-oriented language. However, in the C language, a single root hierarchy is not possible.	Java is also an object-oriented language. However, everything (except fundamental types) is an object in Java. It is a single root hierarchy as everything gets derived from java.lang.Object.

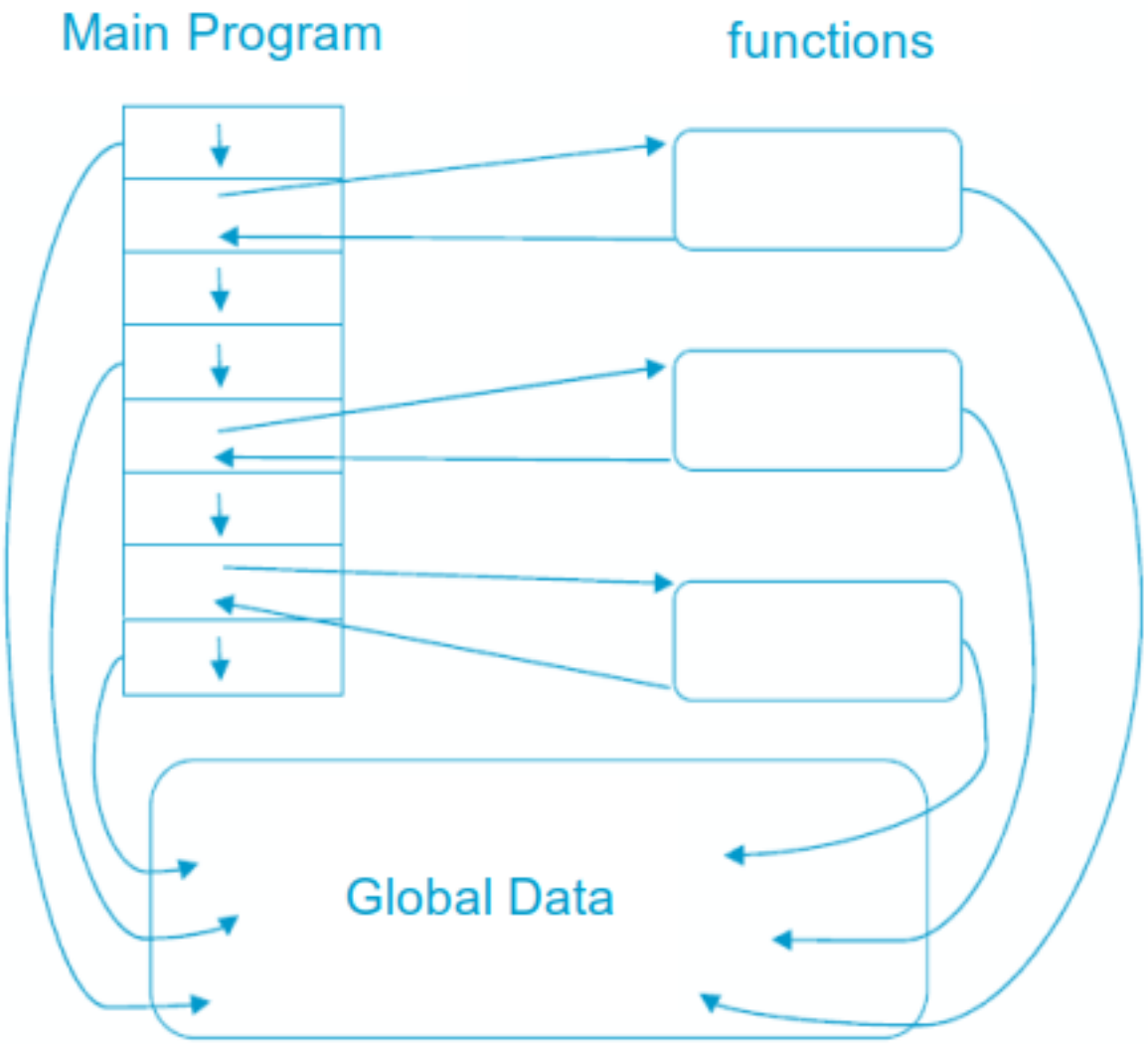
0.116 What is Object Orientation? (Review)

0.117 Procedural Programming

- Pascal, C, Basic, Fortran and similar traditional languages are procedural
  - Each statement tells the computer to do something
- The emphasis is on doing things
  - Functions
- A program is divided into functions
  - Each function has a clearly defined purpose and interface

---

0.118 Procedural Programming



0.119 Problems with Procedural Programming

- **Data Is undervalued**
- Data is, after all, the reason for a program's existence. The important parts of a program **are not functions** that display the data or functions that checks for correct input; they are **data**
- Procedural programs don't model the real world very well. **The real world does not consist of functions**
- **Global data** can be **corrupted** by functions that have no business changing it

- To add new data items, all the functions that access data must be modified so that they can also access these new items
- Creating **new** data types is **difficult**

0.120 Besides

- It is also possible to write good programs by using procedural programming (C programs).

- But object-oriented programming offers programmers many advantages, enables them to write high-quality programs

---

0.121 Object-Oriented Programming

The fundamental idea behind object-oriented programming:

- The **real world** consists of objects. Computer programs may contain computer world representations of the things (objects) that constitute the solutions of real world problems.

- Real world objects have two parts:
  - **Properties** (or state: characteristics that can change),

  - **Behavior** (or abilities: things they can do).
- To solve a programming problem in an object-oriented language,the programmer no longer asks how the problem will be divided into functions, but **how it will be divided into objects**.

- The emphasis is on **data**

0.122 Object-Oriented Programming

- **What kinds of things become objects in object-oriented programs?**
  - **Human entities**: Employees, customers, salespeople,worker, manager
  - **Graphics program**: Point, line, square, circle, …
  - **Mathematics**: Complex numbers, matrix
  - **Computer user environment**: Windows, menus, buttons
  - **Data-storage constructs**: Customized arrays, stacks, linked lists

---

0.123 OOP: Encapsulation and Data Hiding

- Thinking in terms of objects rather than functions
- Close match between **objects** in the **programming** sense and **objects in the real world**
- Both data and the functions that operate on that data are combined into a single program entity
- **Data** represent the **properties** (state), and **functions** represent the **behavior** of an object. Data and its functions are said to be **encapsulated** into a single entity
- An object's functions, called member functions in Java typically provide the only way to access its data. The data is **hidden**, so it is safe from accidental alteration.

0.124 OOP: Encapsulation and Data Hiding

- **Encapsulation** and **data hiding** are key terms in the description of object-oriented languages.
- If you want to modify the data in an object, you know exactly what functions to interact with it
  - The member functions in the object.
  - No other functions can access the data: This simplifies writing, debugging, and maintaining the program.

0.125 Example: A Point on the plane

- A Point on a plane has two properties; x-y coordinates.
- Abilities (behavior) of a Point are, moving on the plane, appearing on the screen and disappearing.

- A model for 2 dimensional points with the following parts:
  - Two integer variables `(x,y)` to represent x and y coordinates
  - A function to move the point: `move`
  - A function to print the point on the screen: `print`
  - A function to hide the point: `hide`

0.126 Example: A Point on the plane

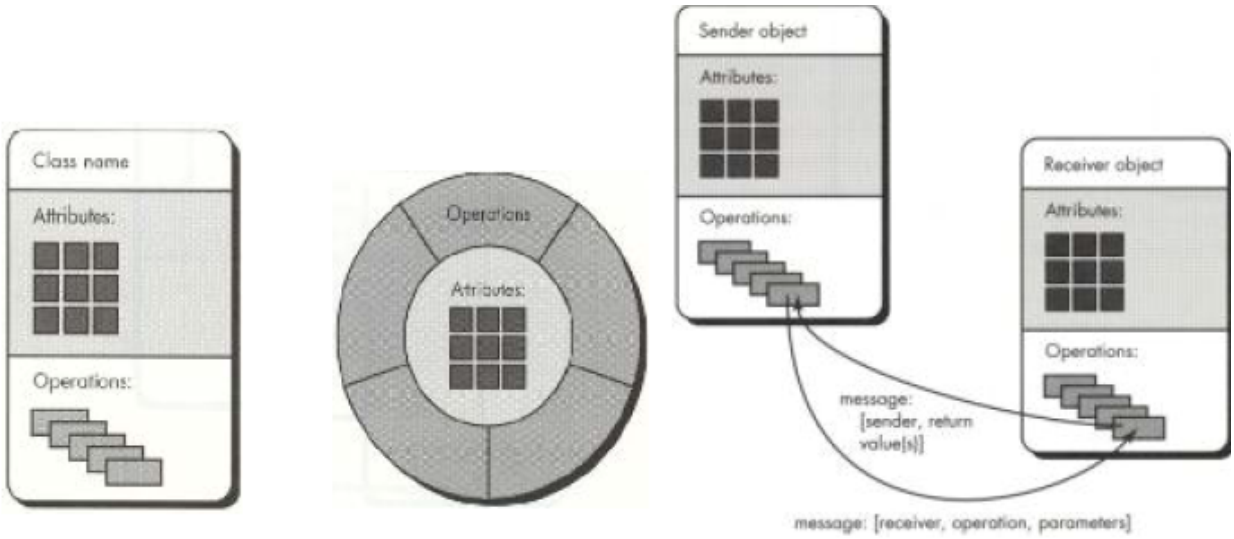
- Once the **model** has been built and tested, it is possible to create many **objects of this model**, in the main program.

```
Point pointOne = new Point(67, 89);
Point pointTwo = new Point(12, 34);

public class Point {
    public int x = 0;
    public int y = 0;
    public Point(int a, int b) {
    x = a;
    y = b;
    }
}
```

0.127 Object Model

A Java program typically consists of a number of objects that communicate with each other by calling one another's member functions.



0.128 OOP vs. Procedural Programming

- Procedural languages still require you to think in terms of the **structure of the computer** rather than the **structure of the problem** you are trying to solve.
- The programmer must establish the association between the **machine model** and the **model of the problem** that is actually being solved.

- The effort required to perform this mapping produces programs that are **difficult to write** and **expensive to maintain**. Because the real world thing and their models on the computer are quite different

---

0.129 Example: Procedural Programming

- Real world thing: **student**
- Computer model: `char *, int, float`
- It is said that the *C* language is **closer to the computer than the problem**.

0.130 OOP vs. Procedural Programming

- The OO approach provides tools for the programmer to **represent elements** in the problem space
- **Objects** are both in the problem space and the solution
- The OO programs are easy to update by adding **new types of objects**
- OOP allows you to describe the problem in terms of the problem, rather than in terms of the computer where the solution will run.

0.131 OOP vs. Procedural Programming

Benefits of the object-oriented programming:

- Readability
- Understandability
- Low probability of errors
- Maintenance
- Reusability
- Teamwork

0.132 OOP vs. Procedural Programming

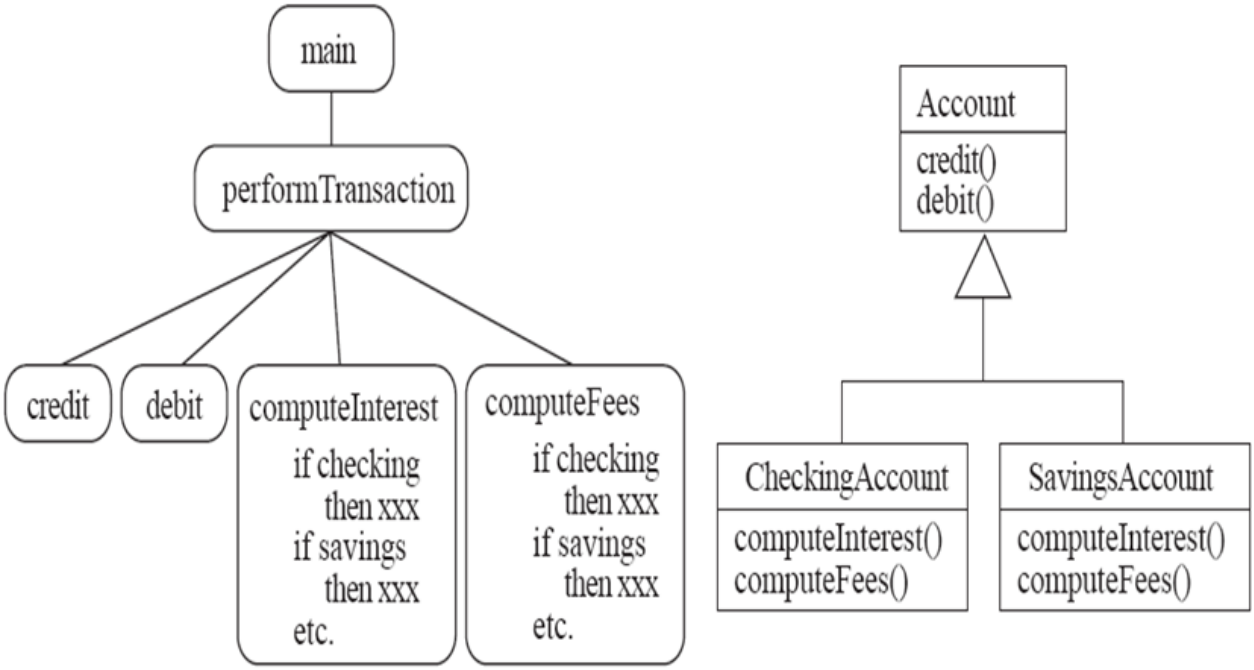
- **Procedural paradigm:**
  - Software is organized around the notion of procedures
  - Procedural abstraction
    - Works as long as the data is simple
- **Adding data abstractions groups together the pieces of data that describe some entity**
  - Helps reduce the system's complexity.
    - Such as Records and structures
- **Object oriented paradigm:**
  - Organizing procedural abstractions in the context of data abstractions

---

0.133 Object Oriented paradigm

- **All computations are performed in the context of objects.**
  - The objects are instances of classes, which:
    - are data abstractions
    - contain procedural abstractions that operate on the objects
  - A running program can be seen as a collection of objects collaborating to perform a given task

0.134 A View of the Two paradigms

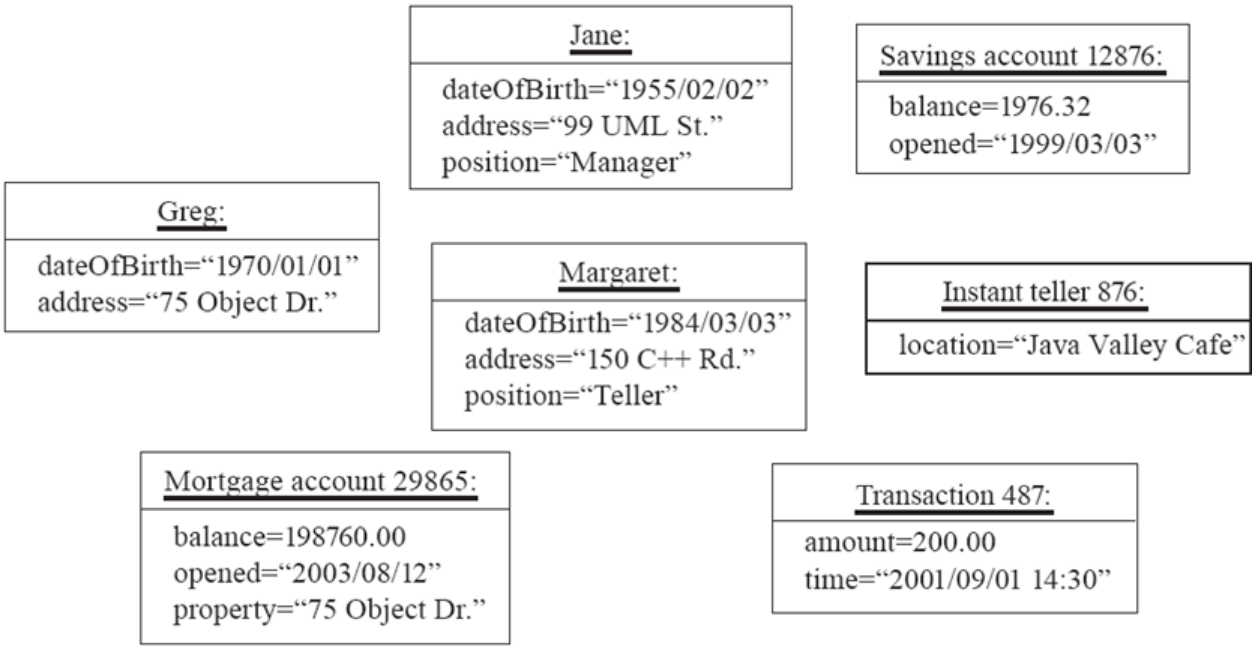


0.135 Classes and Objects

Object

- A chunk of structured data in a running software system
- Has properties
  - Represent its state
- Has behaviour
  - How it acts and reacts
  - May simulate the behaviour of an object in the real world

0.136 Objects: Shown as a UML instance diagram



0.137 Classes

A class:

- A unit of abstraction in an object oriented (OO) program
- Represents similar objects
  - Its instances
- A kind of software module
  - Describes its instances' structure (properties)
  - Contains methods to implement their behaviour

0.138 Is Something a Class or an Instance?

- Something should be a class if it could have instances
- Something should be an instance if it is clearly a single member of the set defined by a class
- **Film**
  - Class; instances are individual films.
- **Reel of Film:**
  - Class; instances are physical reels
- **Film reel with serial number SW19876**
  - Instance of ReelOfFilm

0.139 Is Something a Class or an Instance?

- **Science Fiction**
  - Instance of the class Genre.
- **Science Fiction Film**

  - Class; instances include 'Star Wars'
- **Showing of 'Star Wars' in the Phoenix Cinema at 7 p.m.:**
  - Instance of ShowingOfFilm

0.140 Naming classes

- Use capital letters
  - E.g. `BankAccount` not `bankAccount`
- Use *singular nouns*
- Use the right level of generality
  - E.g. `Municipality`, not `City`
- Make sure the name has only **one** meaning
  - E.g. "bus" has several meanings

0.141 Instance Variables

- **Variables defined inside a class corresponding to data present in each instance**
  - Also called *fields* or *member variables*
  - Attributes
    - Simple data
    - E.g. `name`, `dateOfBirth`
  - Associations
    - Relationships to other important classes
    - E.g. `supervisor`, `coursesTaken`

0.142 Variables vs. Objects

- **A variable**
  - Refers to an object
  - May refer to different objects at different points in time
- **An object can be referred to by several different variables at the same time**
- **Type of a variable**
  - Determines what classes of objects it may contain

0.143 Class variables

- **A class variable's value is shared by all instances of a class.**
  - Also called a *static* variable
  - If one instance sets the value of a class variable, then all the other instances see the same changed value.
  - Class variables are useful for:
    - Default or 'constant' values (e.g. PI)
    - Lookup tables and similar structures

Caution: do not over-use class variables

0.144 Methods, Operations and Polymorphism

- **Operation**
  - A higher-level procedural abstraction that specifies a type of behaviour
  - Independent of any code which implements that behaviour

    * E.g. calculating area (in general)

0.145 Methods, Operations and Polymorphism

- **Method**
  - A procedural abstraction used to implement the behaviour of a class
  - Several different classes can have methods with the same name
    * They implement the same abstract operation in ways suitable to each class
    * E.g. calculating area in a rectangle is done differently from in a circle

---

0.146 Polymorphism

- **A property of object oriented software by which an abstract operation may be performed in different ways in different classes.**
  - Requires that there be *multiple methods* of *the same name*
  - The choice of which one to execute depends on the object that is in a variable
  - Reduces the need for programmers to code many `if-else` or `switch` statements

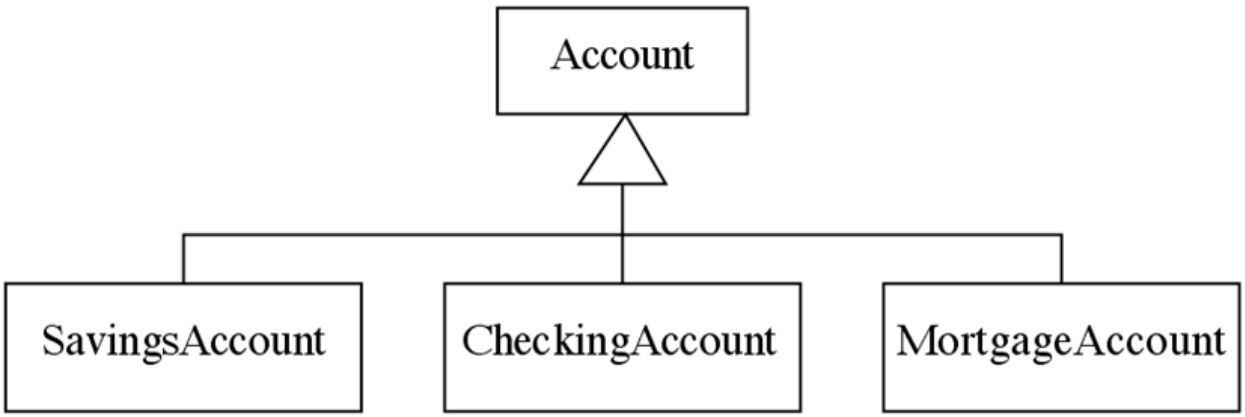
---

0.147 Organizing Classes into Inheritance Hierarchies

- **Superclasses**
  - Contain features common to a set of subclasses
- **Inheritance hierarchies**
  - Show the relationships among superclasses and subclasses
  - A triangle shows a *generalization*
- **Inheritance**
  - The implicit possession by all subclasses of features defined in its superclasses

---

0.148 An Example Inheritance Hierarchy



- **Inheritance**
  - The *implicit* possession by all subclasses of features defined in its superclasses

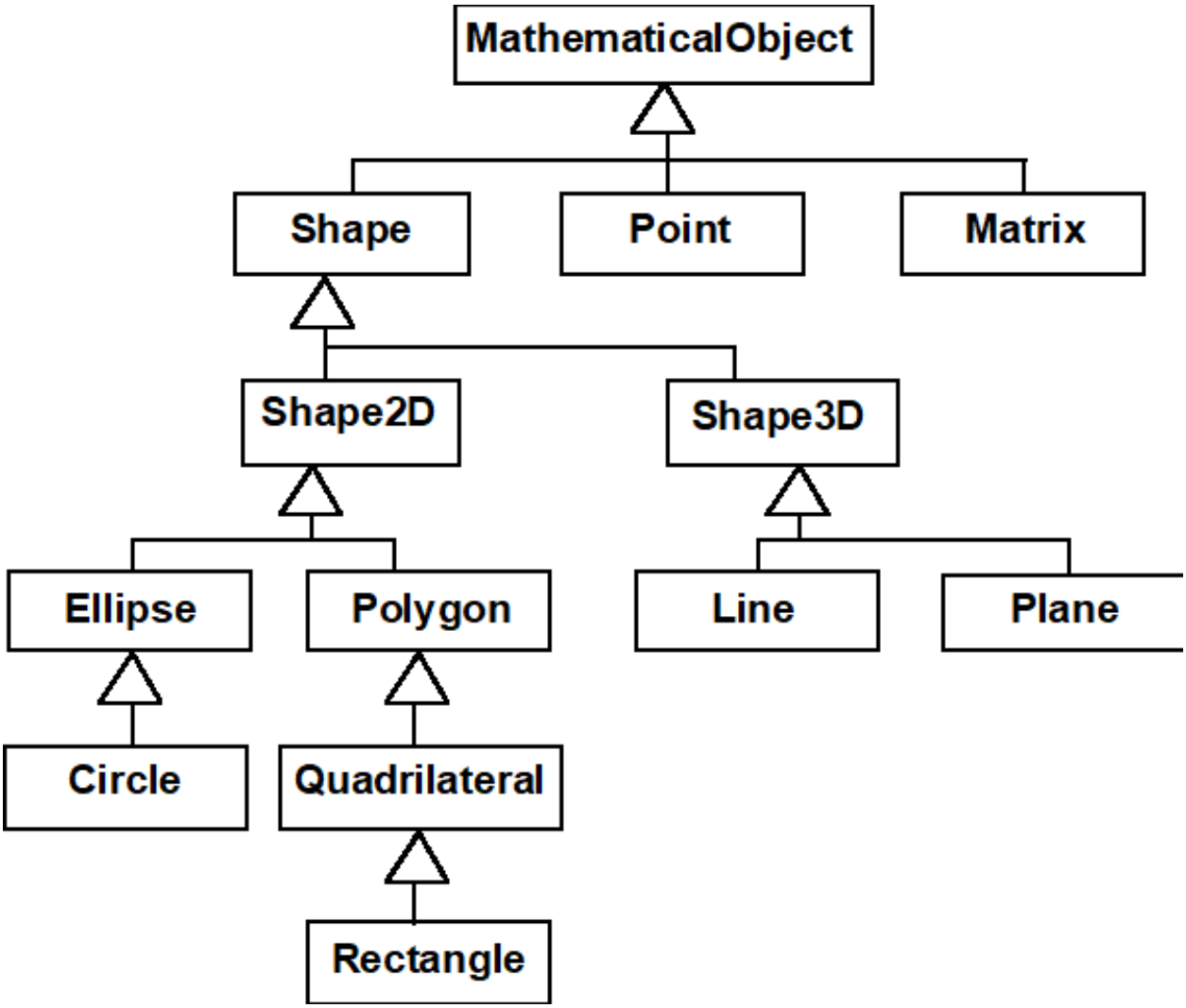
---

0.149 The Is-a Rule

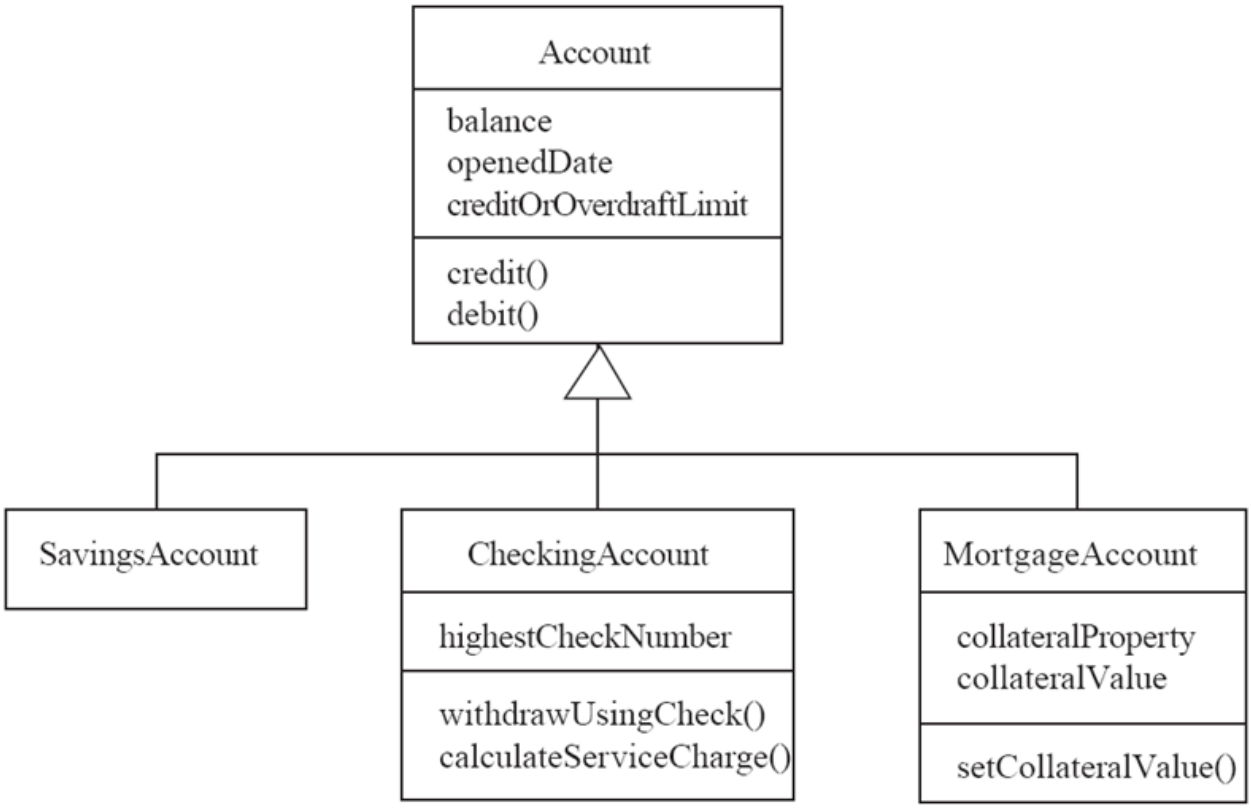
- **Always check generalizations to ensure they obey the isa rule**

  - "A checking account **is an** account"
  - "A village **is a** municipality"
- **Should 'Province' be a subclass of 'Country'?**
  - No, it violates the is-a rule
  - "A province **is a** country" is invalid!

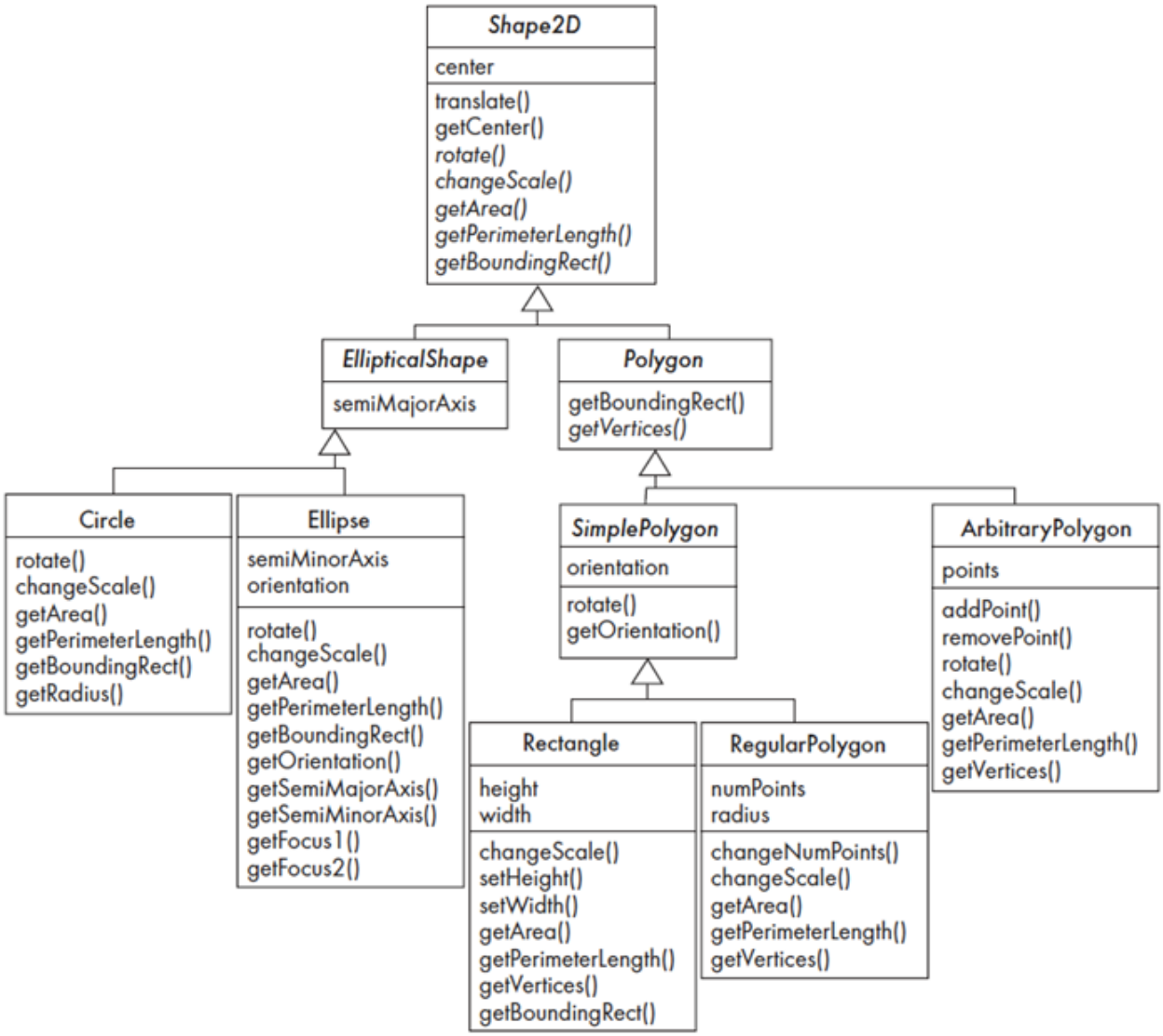
0.150 A possible inheritance hierarchy of mathematical objects



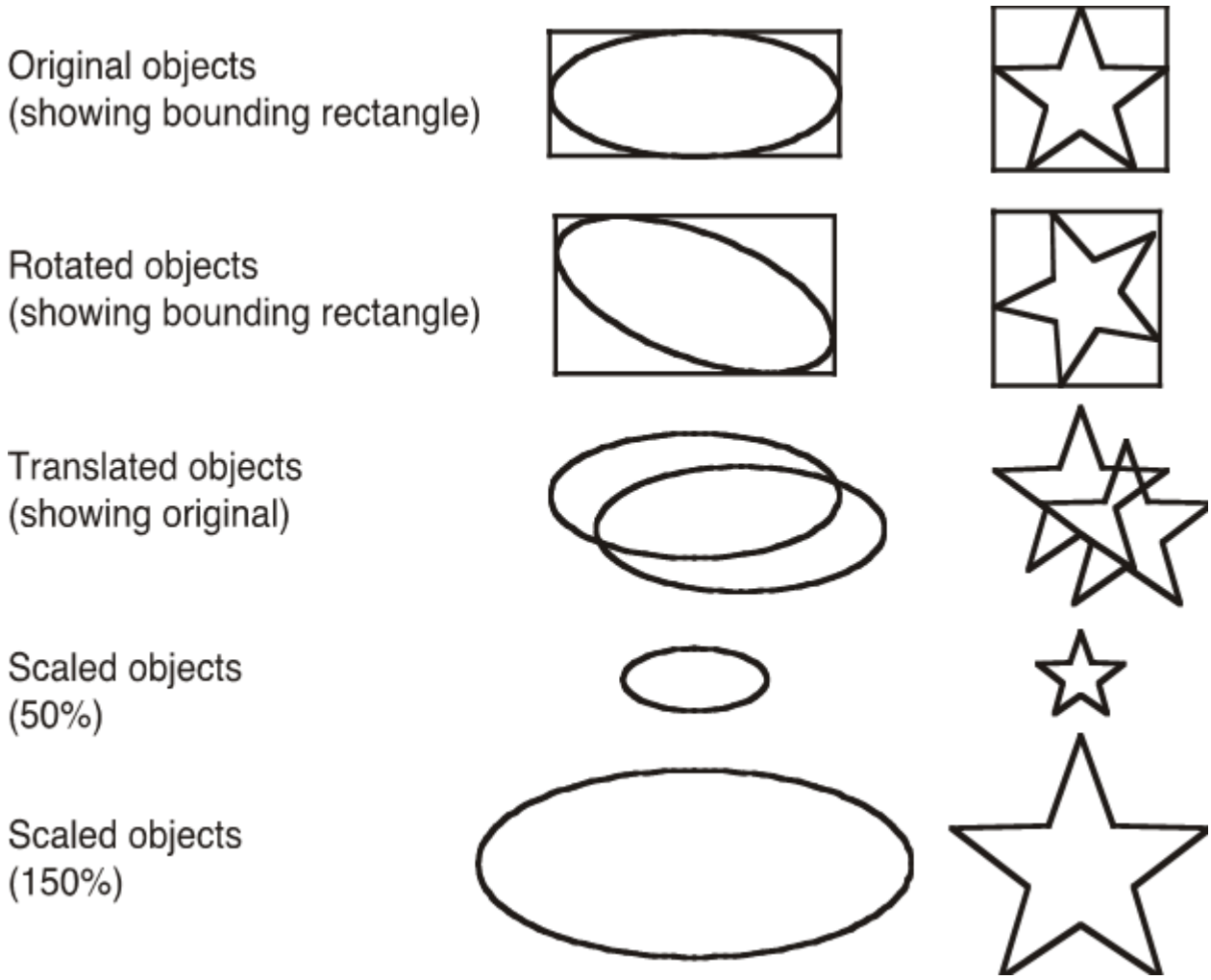
0.151 Make Sure all Inherited Features Make Sense in Subclasses



0.152 Inheritance, Polymorphism and Variables



0.153 Some Operations in the Shape Example



0.154 Abstract Classes and Methods

- **An operation should be declared to exist at the highest class in the hierarchy where it makes sense**
  - The *operation* may be *abstract* (lacking implementation) at that level
  - If so, the class also *must* be abstract
    - No instances can be created
    - The opposite of an abstract class is a *concrete* class
  - If a superclass has an abstract operation then its subclasses at some level must have a concrete method for the operation
    - Leaf classes must have or inherit concrete methods for all operations
    - Leaf classes must be concrete

0.155 Overriding

- A method would be inherited, but a subclass contains a new version instead
  - For extension
    - E.g. `SavingsAccount` might charge an extra fee following every debit
  - For optimization
    - E.g. The `getPerimeterLength` method in `Circle` is much simpler than the one in `Ellipse`

  - For restriction (best to avoid)
    * E.g. `scale(x,y)` would not work in `Circle`

0.156 How a decision is made about which method to run

- If there is a concrete method for the operation in the current class, run that method.
- Otherwise, check in the immediate superclass to see if there is a method there; if so, run it.
- Repeat step 2, looking in successively higher superclasses until a concrete method is found and run.
- If no method is found, then there is an error In Java and C++ the program would not have compiled
  - In Java and C++ the program would not have compiled

0.157 Dynamic binding

- **Occurs when decision about which method to run can only be made at run time**
  - Needed when:
    * A variable is declared to have a superclass as its type, and
    * There is more than one possible polymorphic method that could be run among the type of the variable and its subclasses

---

0.158 Key Terminology

- Abstraction
  - Object $\Longrightarrow$ something in the world
  - Class $\Longrightarrow$ objects
  - Superclass $\Longrightarrow$ subclasses
  - Operation $\Longrightarrow$ methods
  - Attributes and associations $\Longrightarrow$ instance variables
- Modularity
  - Code is divided into classes, and classes into methods
- Encapsulation
  - Details can be hidden in classes
  - This gives rise to *information hiding*:
    * Programmers do not need to know all the details of a class

---

0.159 Basing Software Development on Reusable Technology

0.160 Building on the Experience of Others

Software engineers should avoid re-developing software already developed

- Types of reuse:
  - Reuse of expertise
  - Reuse of standard designs and algorithms
  - Reuse of libraries of classes or procedures
  - Reuse of powerful commands built into languages and operating systems
  - Reuse of frameworks
  - Reuse of complete applications

0.161 Frameworks: Reusable Subsystems

- **A *framework* is reusable software that implements a generic solution to a generalized problem.**
  - It provides common facilities applicable to different application programs. **- *Principle*: Applications that do different, but related, things tend to have similar designs**

0.162 Frameworks to promote reuse

- **A framework is intrinsically incomplete**
  - Certain classes or methods are used by the framework, but are missing (*slots*)
  - Some functionality is optional
    - Allowance is made for developer to provide it (*hooks* or *extension points*)
  - Developers use the services that the framework provides
    - Taken together the services are called the Application Program Interface (*API*)

0.163 Object-oriented frameworks

- **In the object oriented paradigm, a framework is composed of a library of classes.**
  - The API is defined by the set of all public methods of these classes.
  - Some of the classes will normally be abstract and there are often many Interfaces
- **Example:**
  - A framework for payroll management
  - A framework for frequent buyer clubs
  - A framework for university registration
  - A framework for e-commerce web sites

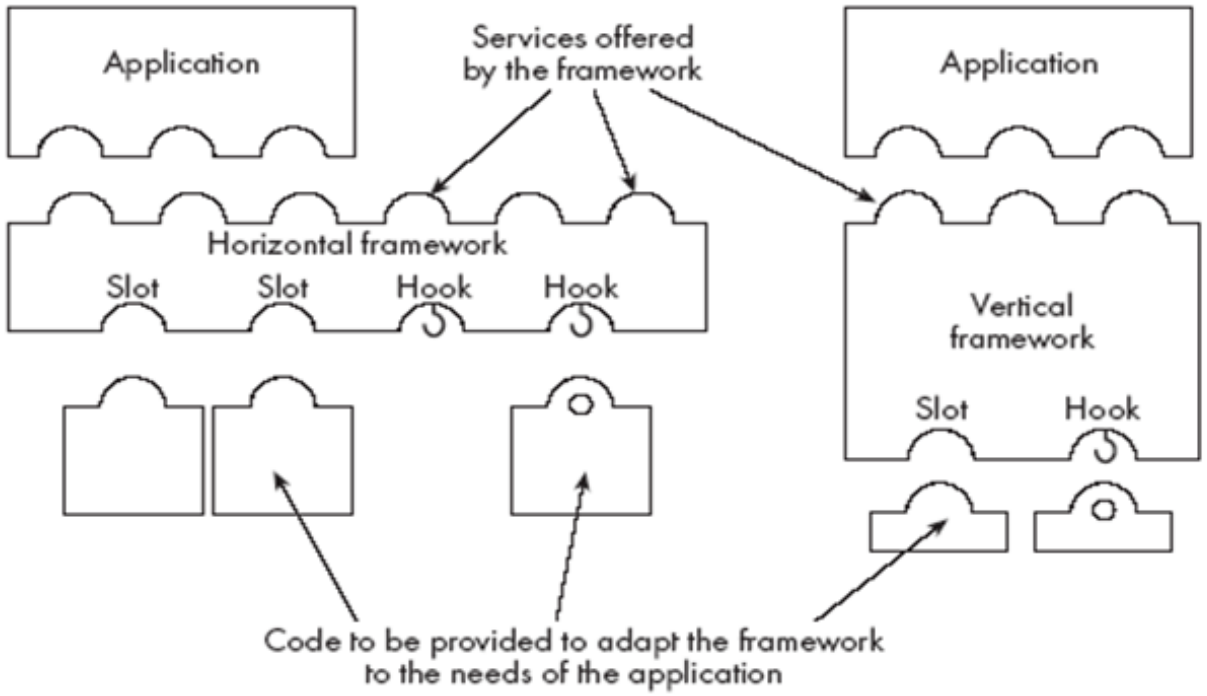
0.164 Frameworks and product lines

- A product line (or product family) is a set of products built on a common base of technology.
  - The various products in the product line have different features to satisfy different markets
  - The software common to all products in included in a framework
  - Each product is produced by filling the available hooks and slots
    - E.g. software products offering “demo”, “lite” or “pro” versions

---

0.165 Types of frameworks

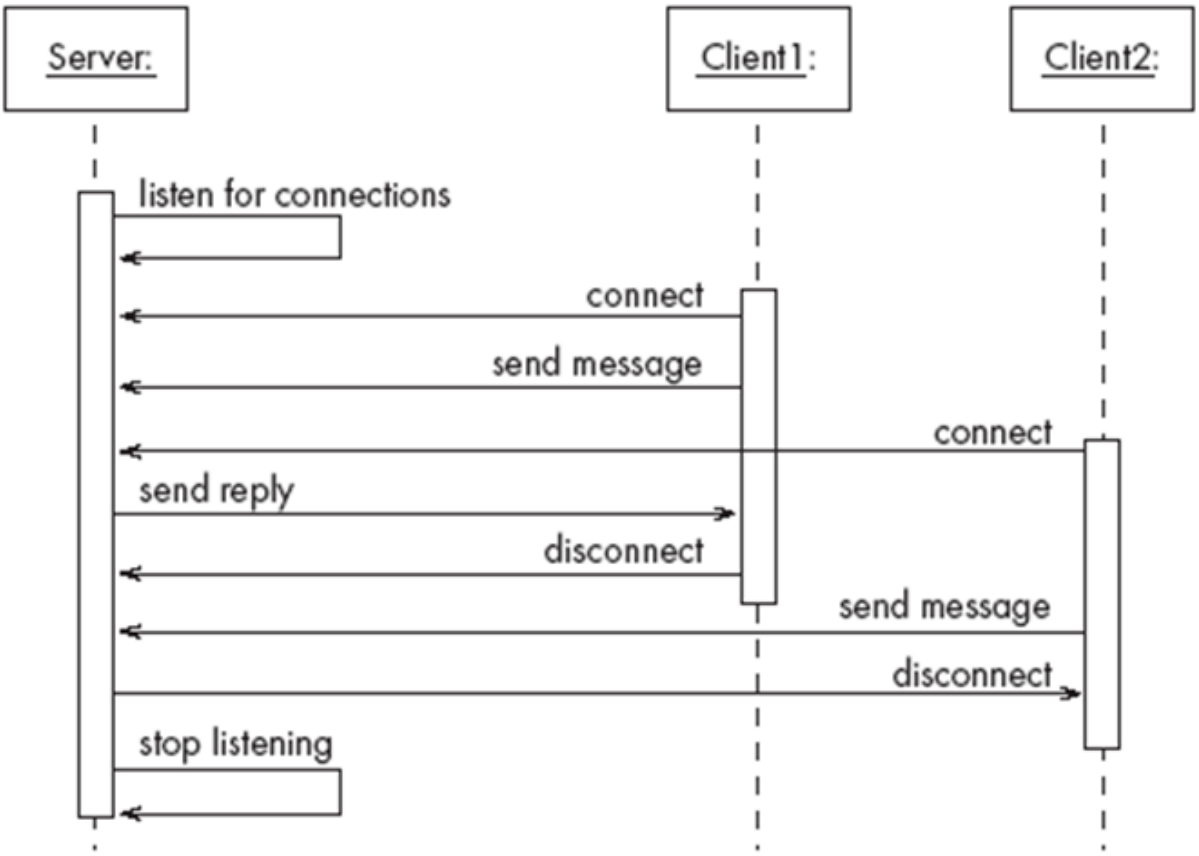
- A *horizontal* framework provides general application facilities that a large number of applications can use
- A *vertical* framework (*application framework*) is more ‘complete’ but still needs some slots to be filled to adapt it to specific application needs



0.166 The Client-Server Architecture

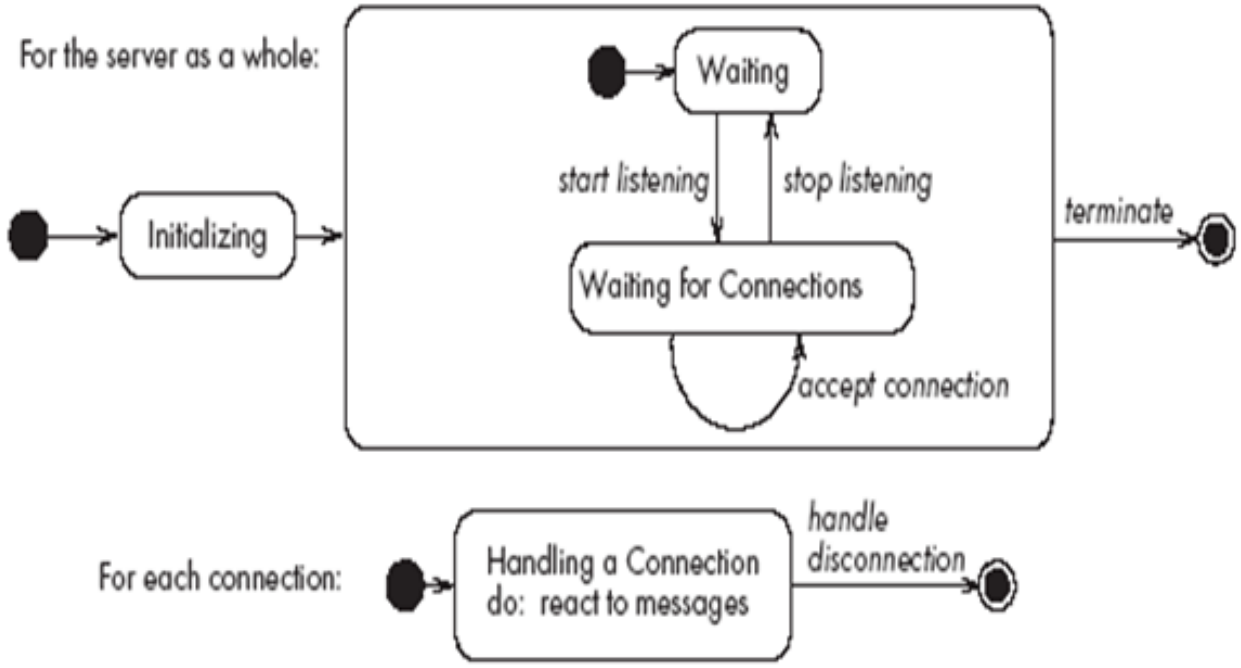
- **A distributed system is a system in which:**
  - computations are performed by *separate* programs
  - … normally running on separate pieces of hardware
  - … that *co-operate* to perform the task of the system.
- **Server:**
  - A program that *provides a service* for other programs that connect to it using a communication channel
- **Client**
  - A program that accesses a server (or several servers) to *obtain services*
  - A server may be accessed by many clients simultaneously

0.167 Example of client-server systems



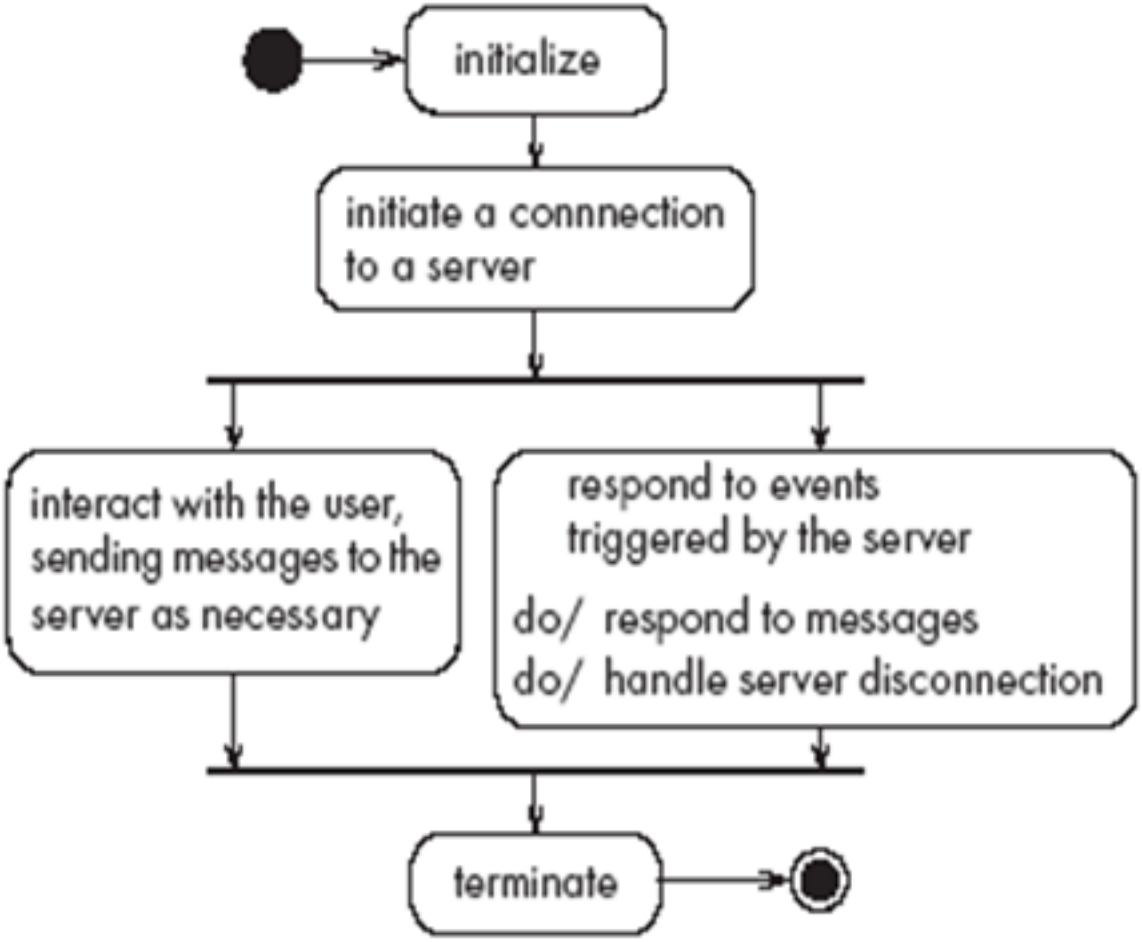
0.168 Activities of a server

- Initializes itself
- Starts listening for clients
- Handles the following types of events originating from clients
  - accepts connections
  - responds to messages
  - handles client disconnection
- May stop listening
- Must cleanly terminate

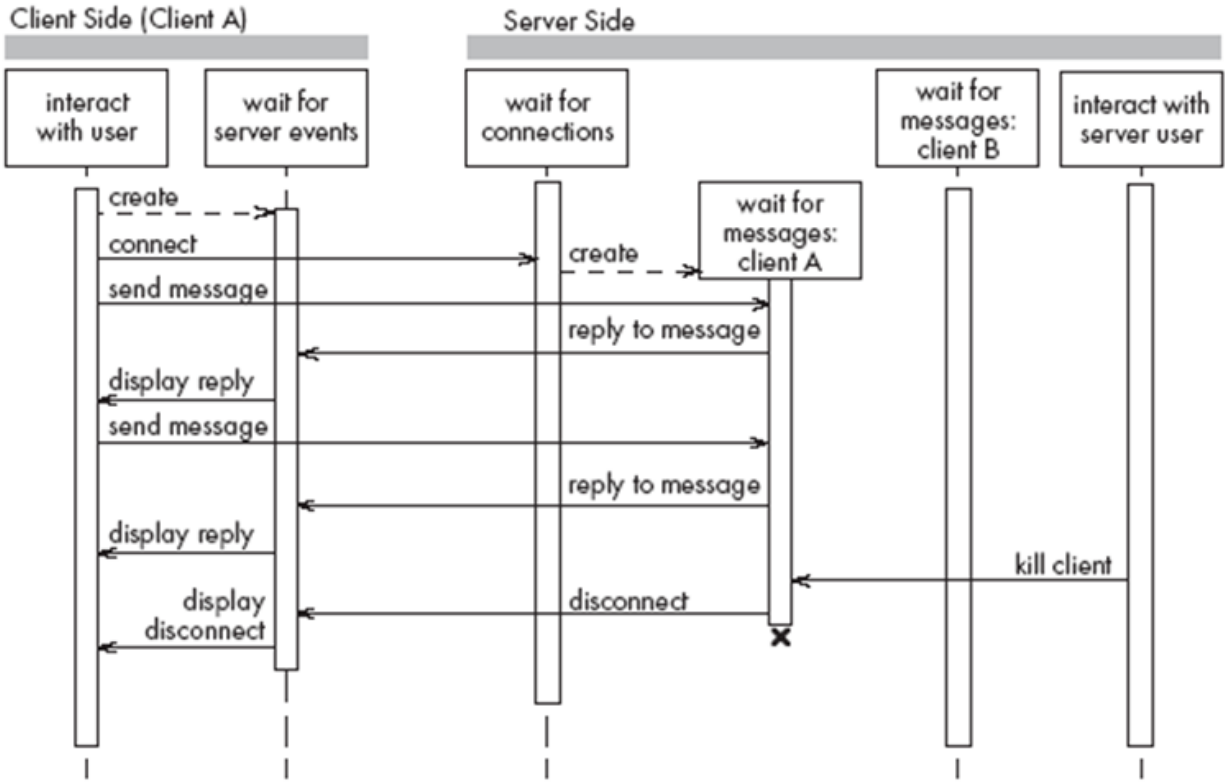


0.169 Activities of a client

- Initializes itself
- Initiates a connection
- Sends messages
- Handles the following types of events originating from the server
  - responds to messages
  - handles server disconnection
- Must cleanly terminate

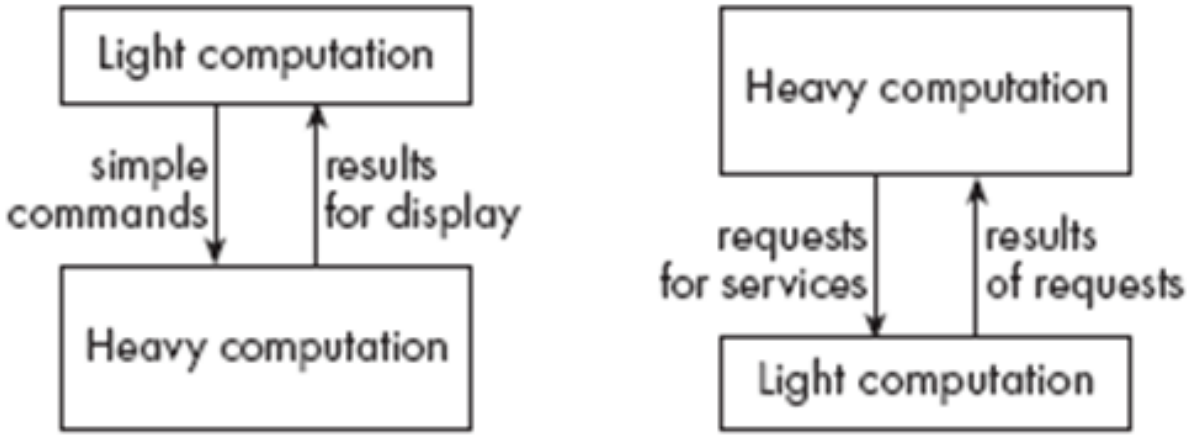

initialize
initiate a connnection
to a server
interact with the user,
sending messages to the
server as necessary
respond to events
triggered by the server
do/ respond to messages
do/ handle server disconnection
terminate

0.170 Threads in a client-server system



0.171 Thin- versus fat-client systems

- **Thin-client system (a)**
  - Client is made as small as possible
  - Most of the work is done in the server.
  - Client easy to download over the network
- **Fat-client system (b)**
  - As much work as possible is delegated to the clients.
  - Server can handle more clients



0.172 Communications protocols

- The messages the client sends to the server form a language.
  - The server has to be programmed to understand that language.
- The messages the server sends to the client also form a language.
  - The client has to be programmed to understand that language.
- When a client and server are communicating, they are in effect having a conversation using these two languages
- The two languages and the rules of the conversation, taken together, are called the protocol

---

0.173 Tasks to perform to develop client-server applications

- Design the primary work to be performed by both client and server
- Design how the work will be distributed
- Design the details of the set of messages that will be sent
- Design the mechanism for
  - Initializing
  - Handling connections
  - Sending and receiving messages
  - Terminating

0.174 Advantages of client-server systems

- The work can be *distributed* among different machines
- The clients can access the server's functionality from a *distance*
- The client and server can be designed separately
- They can both be *simpler*
- There is a choice about where to keep data:
  - All the *data can be kept centrally* at the server
  - Data *can be distributed* among many different clients or servers
- The server can be accessed *simultaneously* by many clients
- *Competing clients can be written* to communicate with the same server, and vice-versa

---

0.175 Technology Needed to Build Client-Server Systems

- Internet Protocol (IP)
  - Route messages from one computer to another
  - Long messages are normally split up into small pieces

- Transmission Control Protocol (TCP)
  - Handles connections between two computers
  - Computers can then exchange many IP messages over a connection
  - Assures that the messages have been satisfactorily received
- A host has an IP address and a host name
  - Several servers can run on the same host.
  - Each server is identified by a port number (0 to 65535).
  - To initiate communication with a server, a client must know both the host name and the port number

---

0.176 Establishing a connection in Java

- The `java.net` package

  - Permits the creation of a TCP/IP connection between two applications
- Before a connection can be established, the server must start listening to one of the ports:

```
ServerSocket serverSocket = new ServerSocket(port);
Socket clientSocket = serverSocket.accept();
```

- For a client to connect to a server:

```
Socket clientSocket= new Socket(host, port);
```

0.177 Exchanging information in Java

- Each program uses an instance of
  - InputStream to receive messages from the other program
  - OutputStream to send messages to the other program
  - These are found in package java.io

```
output = clientSocket.getOutputStream();
```

```
input = clientSocket.getInputStream();
```

0.178 Sending and receiving messages

- without any filters (raw bytes)

```
output.write(msg);
msg = input.read();
```

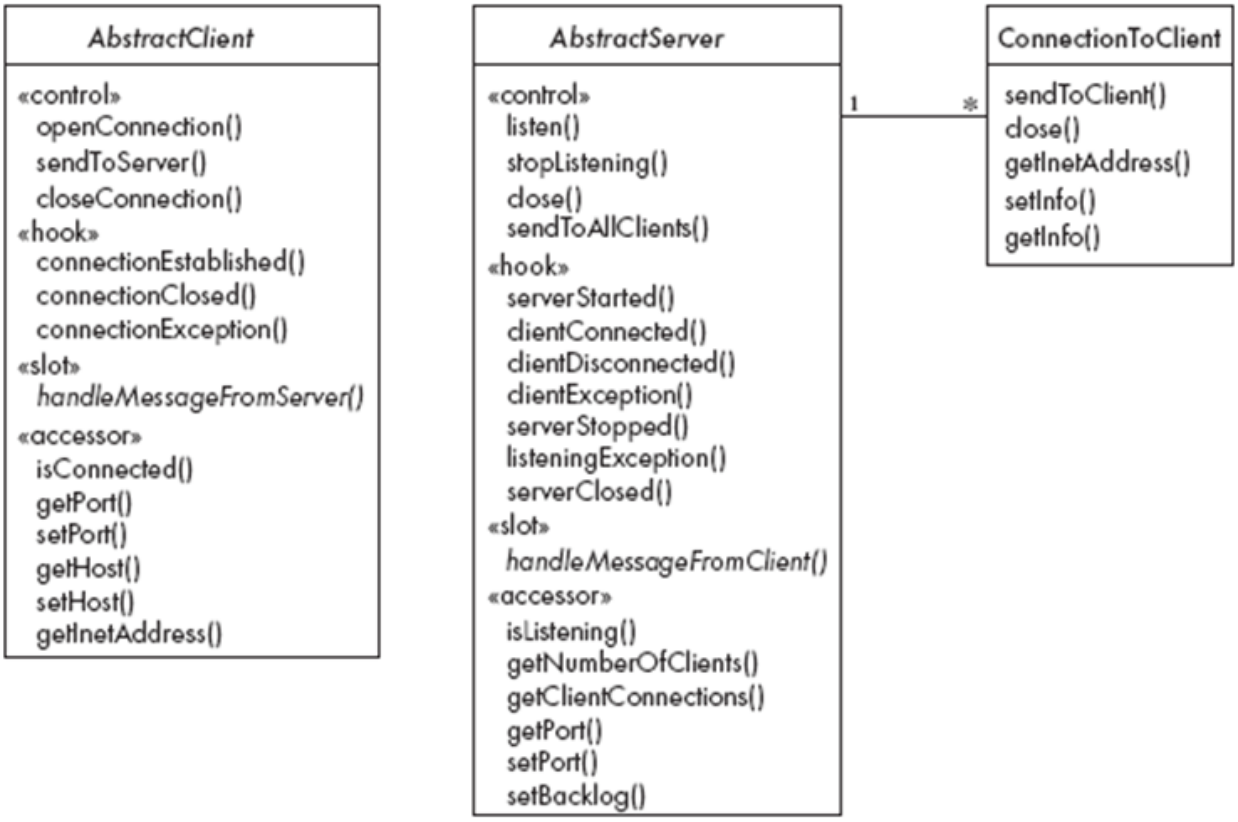
- or using DataInputStream / DataOutputStream filters

```
output.writeDouble(msg);
msg = input.readDouble();
```

- or using ObjectInputStream / ObjectOutputStream filters

```
output.writeObject(msg);
msg = input.readObject();
```

0.179 The Object Client-Server Framework (OCSF)



0.180 Using OCSF

- **Software engineers using OCSF never modify its three classes**
- **They:**
  - Create subclasses of the abstract classes in the framework
  - Call public methods that are provided by the framework
  - Override certain slot and hook methods (explicitly designed to be overridden)

0.181 The Client Side

- Consists of a single class: AbstractClient
  - Must be subclassed
    - Any subclass must provide an implementation for handleMessageFromServer
      - Takes appropriate action when a message is received from a server
- Implements the Runnable interface
  - Has a run method which
    - Contains a loop that executes for the lifetime of the thread

---

0.182 The public interface of AbstractClient

- **Controlling methods:**
  - openConnection
  - closeConnection

  - sendToServer
- **Accessing methods:**
  - isConnected
  - getHost
  - setHost
  - getPort
  - setPort
  - getInetAddress

0.183 The callback methods of AbstractClient

- **Methods that may be overridden:**
  - connectionEstablished
  - connectionClosed
- **Method that must be implemented:**
  - handleMessageFromServer

0.184 Using AbstractClient

- Create a subclass of **AbstractClient**
- Implement **handleMessageFromServer** slot method
- Write code that:
  - Creates an instance of the new subclass
  - Calls **openConnection**
  - Sends messages to the server using the **sendToServer** service method
- Implement the **connectionClosed** callback
- Implement the **connectionException** callback

0.185 Internals of AbstractClient

- Instance variables:
  - A **Socket** which keeps all the information about the connection to the server
  - Two streams, an **ObjectOutputStream** and an **ObjectInputStream**
  - A **Thread** that runs using **AbstractClient's** run method
  - Two variables storing the host and port of the server

0.186 The Server Side

- Two classes:
  - One for the thread which listens for new connections (**AbstractServer**)
  - One for the threads that handle the connections to clients (**ConnectionToClient**)

0.187 The public interface of AbstractServer

- **Controlling methods:**
  - listen
  - stopListening
  - close
  - sendToAllClients

- **Accessing methods:**
  - isListening
  - getClientConnections
  - getPort
  - setPort
  - setBacklog

0.188 The callback methods of AbstractServer

- **Methods that may be overridden:**
  - serverStarted
  - clientConnected
  - clientDisconnected
  - clientException
  - serverStopped
  - listeningException
  - serverClosed
- **Method that must be implemented:**
  - handleMessageFromClient

0.189 The public interface of ConnectionToClient

- **Controlling methods:**
  - sendToClient
  - close
- **Accessing methods:**
  - getInetAddress
  - setInfo
  - getInfo

0.190 Using AbstractServer and ConnectionToClient

- Create a subclass of **AbstractServer**
- Implement the slot method **handleMessageFromClient**
- Write code that:
  - Creates an instance of the subclass of **AbstractServer**
  - Calls the **listen** method
  - Sends messages to clients, using:
    - the **getClientConnections** and **sendToClient** service methods
    - or **sendToAllClients**
- Implement one or more of the other callback methods

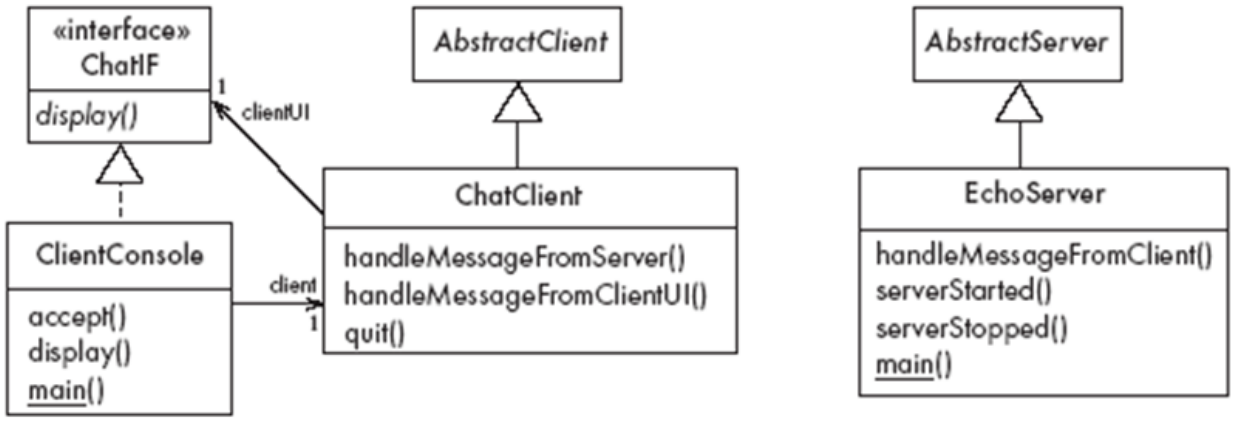
0.191 Internals of AbstractServer and ConnectionToClient

- The **setInfo** and **getInfo** methods make use of a Java class called **HashMap**
- Many methods in the server side are **synchronized**
- The collection of instances of **ConnectionToClient** is stored using a special class called **ThreadGroup**
- The server must pause from listening every *500ms* to see if the **stopListening** method has been called

- if not, then it resumes listening immediately

0.192 An Instant Messaging Application: SimpleChat

- ClientConsole can eventually be replaced by ClientGUI



0.193 The server

- **EchoServer** is a subclass of **AbstractServer**
  - The **main** method creates a new instance and starts it
    - It listens for clients and handles connections until the server is stopped
- The three callback methods just print out a message to the user
  - **handleMessageFromClient**, **serverStarted** and **serverStopped**
- The slot method **handleMessageFromClient** calls **sendToAllClients**
  - This echoes any messages

0.194 Key code in EchoServer

```
public void handleMessageFromClient
  (Object msg, ConnectionToClient client)
{
   System.out.println(
     "Message received: "
     + msg + " from " + client);
   this.sendToAllClients(msg);
}
```

0.195 The client

- **When the client program starts, it creates instances of two classes:**
  - **ChatClient**
    - A subclass of AbstractClient
    - Overrides handleMessageFromServer
      - This calls the display method of the user interface
  - **ClientConsole**
    - User interface class that implements the interface **ChatIF**
      - Hence implements **display** which outputs to the console

    * Accepts user input by calling **accept** in its **run** method
    * Sends all user input to the **ChatClient** by calling its **handleMessageFromClientUI**
        · This, in turn, calls **sendToServer**

0.196 Key code in ChatClient

```
public void handleMessageFromClientUI(
  String message)
{
  try
  {
    sendToServer(message);
  }
  catch(IOException e)
  {
    clientUI.display (
       "Could not send message. " +
       "Terminating client.");
    quit();
  }
}
```

0.197 Key code in ChatClient

```
public void handleMessageFromServer(Object msg)
{
  clientUI.display(msg.toString());
}
```

0.198 Risks when reusing technology

- **Poor quality reusable components**
  - Ensure that the developers of the reusable technology:
    * follow good software engineering practices
    * are willing to provide active support
- **Compatibility not maintained**
  - Avoid obscure features
  - Only re-use technology that others are also re-using

0.199 Risks when developing reusable technology

- **Investment uncertainty**
  - Plan the development of the reusable technology, just as if it was a product for a client
- **The "not invented here syndrome"**
  - Build confidence in the reusable technology by:
    * Guaranteeing support
    * Ensuring it is of high quality
    * Responding to the needs of its users

0.200 Risks when developing reusable technology

- **Competition**
  - The reusable technology must be as useful and as high quality as possible
- **Divergence** (tendency of various groups to change technology in different ways)
  - Design it to be general enough, test it and review it in advance

---

0.201 Risks when adopting a client-server approach

- **Security**
  - Security is a big problem with no perfect solutions: consider the use of encryption, firewalls, ...
- **Need for adaptive maintenance**
  - Ensure that all software is forward and backward compatible with other versions of clients and servers

---

0.202 References

https://www.site.uottawa.ca/~tcl/seg2105/

https://cruise.umple.org/index.shtml

https://cruise.umple.org/umple/GettingStarted.html

Sanem Sarıel Associate Professor, PhD BT503 Application Development with Java (Kemerburgaz University 2013-2015)[5]

How To Define The Project Scope The Foolproof Way[6]

Java Tutorials - OOP Concepts | Encapsulation | Abstraction | Inheritance | Polymorphism[7]

[5] https://web.itu.edu.tr/sariel/teaching.php
[6] https://medium.com/@ayush_90732/how-to-define-the-project-scope-the-foolproof-way-782b239db2bc
[7] http://www.btechsmartclass.com/java/java-oop-concepts.html