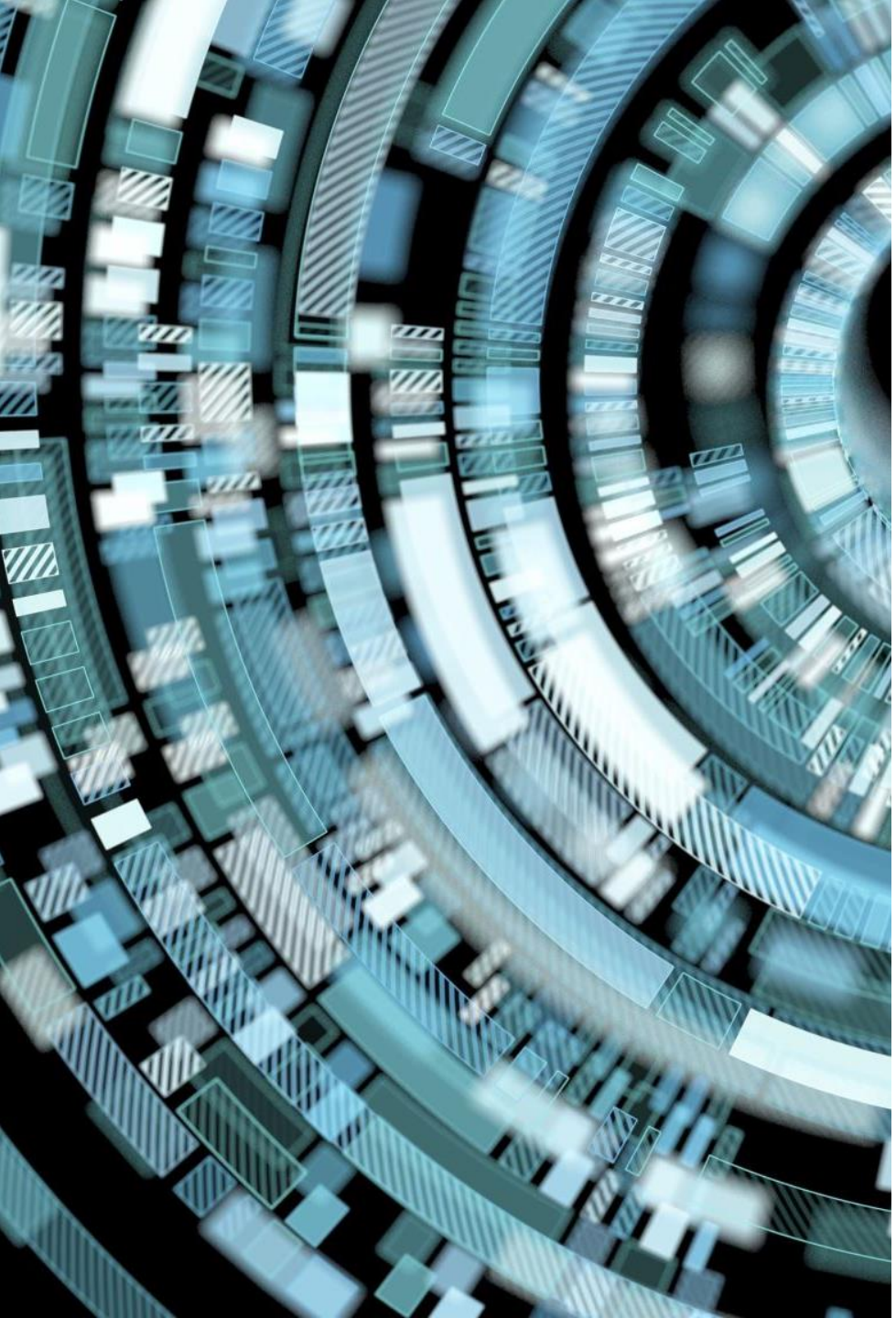
Computer System Design & Application

计算机系统设计与应用A

陶伊达 （TAO Yida）
taoyd@sustech.edu.cn



Lecture 1

- Course introduction
- Computer system & programs
- Java overview, JVM, and Virtualization
- Software design principles
- Object-Oriented Programming Concepts

Course Logistics

- Course website: Sakai https://sakai.sustech.edu.cn/portal/directtool/68836b5e-4e66-4c92-906e-47f58df77a05/
- Slides and other resources will all be uploaded here.

- Office hours: Monday

  10:30 – 11:30 am

College of Engineering South Building, 441B

计算机系统设计与应用A (2022秋)
Computer System Design and Application (Java2 for short)

Lecturer: 陶伊达, taoyd@sustech.edu.cn

理论课 （1-16周）(QQ群：761890313)

每周二下午，7-8节，三教208 （SA：邹若彤）

实验课 （1-16周）

实验1组 周三下午5-6节 三教510 SA：吴笑丰、易翔 QQ群：649100024
实验2组 周三上午3-4节 三教503 SA：邱逸伦，徐驰 QQ群：810309063

Online Course Arrangement

腾讯会议南科大教育版管理平台

课程号：CS209A001　　授课人：陶伊达　　授课班级：计算机系统设计及应用A001

自建课次　修改教学方式

	上课时间	授课人	教学平台	上课链接	辅助教学方式	发布状态	操作	课次来源
	09月06日 周二 7-8节课	陶伊达	腾讯会议南科大教育版	https://meeting.tencent.com/dm/Ftrew3MmWCrz		已发布		同步教务课次
	09月13日 周二 7-8节课	陶伊达	腾讯会议南科大教育版			已发布（会议室待开放）		同步教务课次
	09月20日 周二 7-8节课	陶伊达	腾讯会议南科大教育版			已发布（会议室待开放）		同步教务课次

Course Objective

- An understanding of new topics in programming and computer application system design
- An understanding of design principles and good practices in software application design & development
- An understanding of advanced programming topics and skills useful for scientific & engineering students
- Using Java to solve practical problems efficiently and effectively

Topics covered

Principles

- OOP
- Design patterns
- Functional programming
- Reusable software
- Software engineering

……

Utilities

- Exception handling
- Generic collections
- Lambdas & Streams
- Annotation
- Testing

……

Functionalities

- File I/O
- GUI
- Networking
- Reflection
- Web development

……

Applications

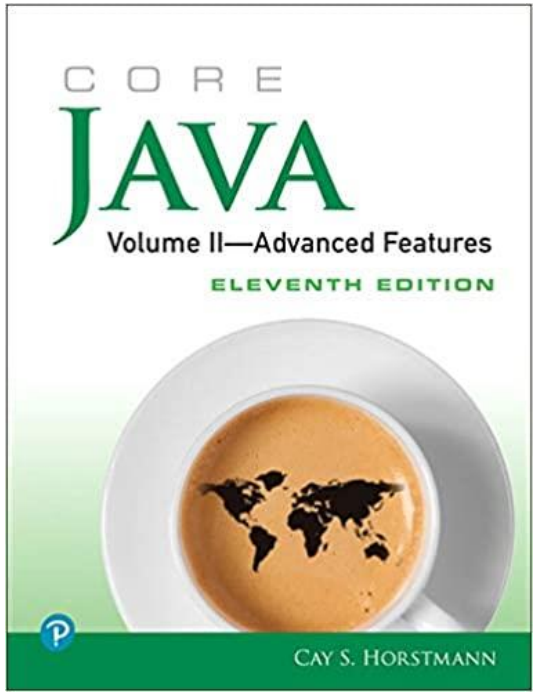
- Text scraping and processing
- Data analytics and visualization
- Web applications & services

……

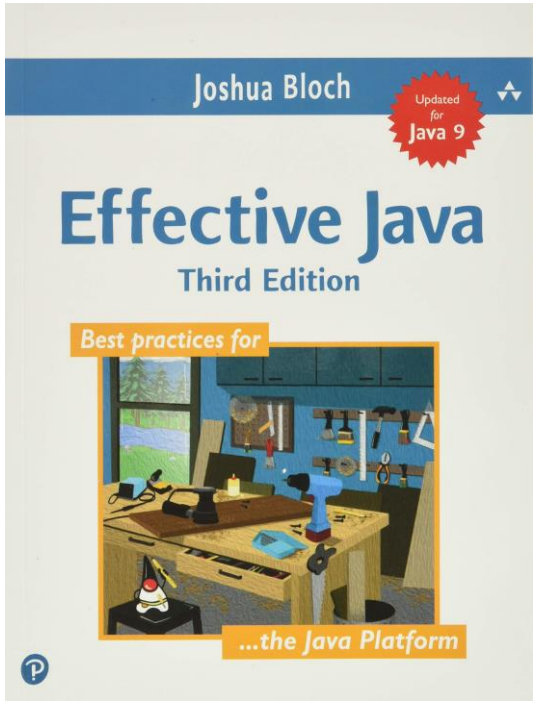
Syllabus
(Negotiable)

- Lecture 1: Computing overview, JVM, Software Design Principles
- Lecture 2: Generics, ADT, Collections
- Lecture 3: Functional programming, Lambda
- Lecture 4: Java 8 Stream API
- Lecture 5: I/O Streams, Encoding
- Lecture 6: Serialization, File I/O, Exception Handling
- Lecture 7: Concurrency, Multithreading
- Lecture 8: Network Programming
- Lecture 9: GUI Intro, JavaFX
- Lecture 10: Reflection, Annotation
- Lecture 11: JUnit Testing
- Lecture 12: Java EE, Servlet
- Lecture 13: Spring, Spring Boot
- Lecture 14: Design Patterns
- Lecture 15: Miscellaneous
- Lecture 16: Project Presentation

Reference Books



Core Java Volume II – Advanced Features
Cay S. Horstmann



Effective Java
Joshua Bloch

Coursework & Grading Policy

	Score	Description
Labs	15%	Attendance Lab practices (+0.1 points for submitting lab practice onsite, max +1)
Assignments	25%	2 assignments Assignment 1: release at week 4 and due at week 7 Assignment 2: release at week 8 and due at week 11
Project	20%	Released no later than week 8 Team: Preferably 2 people +1 for submitting the final project at week 15 +2 (max) for presenting at week 16 lecture
Standardization	4%	Version control (git) Coding styles / coding convention
Quiz	6%	Quizzes during lectures
Final Exam	30%	Close-book (Two pieces of A4 cheat sheets allowed) No electronic device

Labs start from the 1st week!

Academic Integrity

From Spring 2022, the plagiarism policy applied by the Computer Science and Engineering department is the following: ↵

↵

*** If an undergraduate assignment is found to be plagiarized, the first time the score of the assignment will be 0.**↵

*** The second time the score of the course will be 0.**↵

*** If a student does not sign the Assignment Declaration Form or cheats in the course, including regular assignments, midterms, final exams, etc., in addition to the grade penalty, the student will not be allowed to enroll in the two CS majors through 1+3, and cannot receive any recommendation for postgraduate admission exam exemption and all other academic awards.**↵

↵

As it may be difficult when two assignments are identical or nearly identical who actually wrote it, the policy will apply to BOTH students, unless one confesses having copied without the knowledge of the other. ↵

- It's OK to work on an assignment with a friend, and think together about the program structure, share ideas and even the global logic. At the time of actually writing the code, you should write it alone.
- It's OK to use in an assignment a piece of code found on the web, as long as you indicate in a comment where it was found and don't claim it as your own work.
- It's OK to help friends debug their programs (you'll probably learn a lot yourself by doing so).
- It's OK to show your code to friends to explain the logic, as long as the friends write their code on their own later.

- It's NOT OK to take the code of a friend, make a few cosmetic changes (comments, some variable names) and pass it as your own work.

Academic Integrity

Please submit the form before the end of the course selection & drop period!

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY | 计算机科学与工程系 Department of Computer Science and Engineering

本科生作业承诺书

本人_____（学号_____）本学期已选修计算机科学与工程系_____课程。本人已阅读并了解《南方科技大学计算机科学与工程系本科生作业抄袭学术不端行为的认定标准及处理办法》制度中关于禁止本科生作业抄袭的相关规定，并承诺自觉遵守其规定。

承诺人：

年　　月　　日

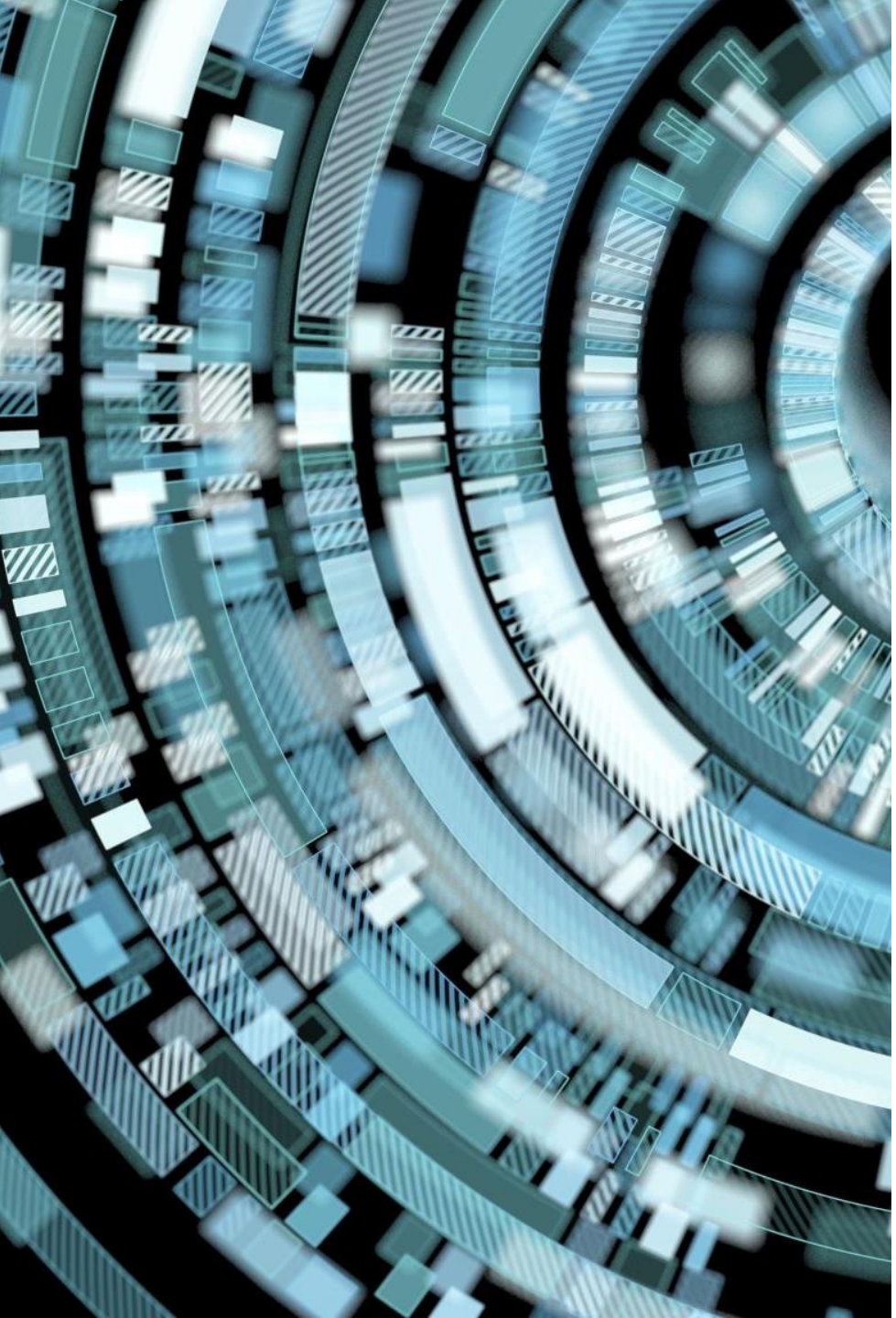
南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY | 计算机科学与工程系 Department of Computer Science and Engineering

Undergraduate Students Assignment Declaration Form

This is_____(student ID:_____, who has enrolled in _____ course, originated the Department of Computer Science and Engineering. I have read and understood the regulations on plagiarism in assignments and theses according to "Regulations on Academic Misconduct in Assignments for Undergraduate Students in the SUSTech Department of Computer Science and Engineering". I promise that I will follow these regulations during the study of this course.

Signature:

Date:



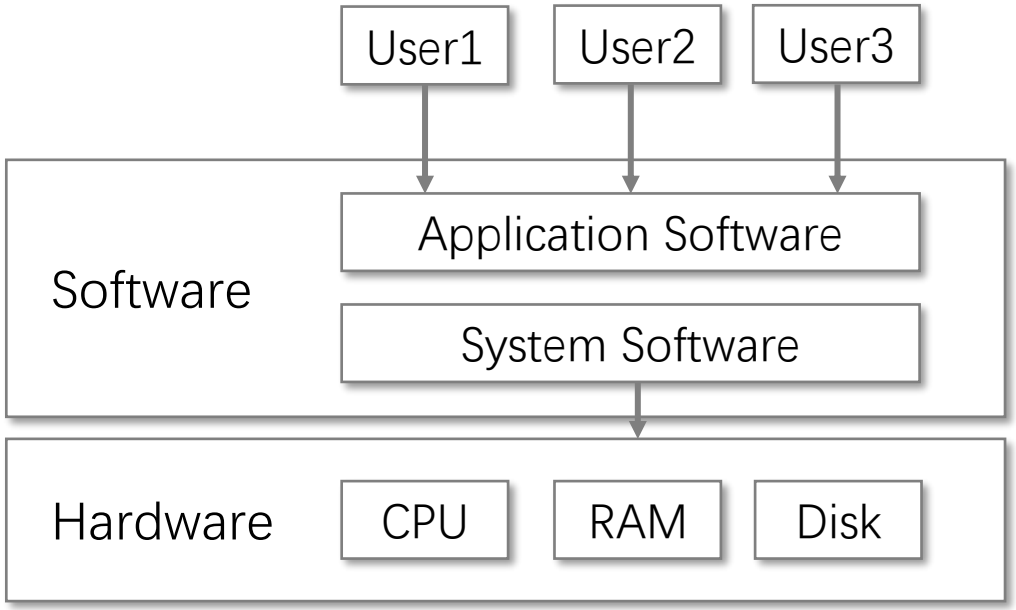
Lecture 1

- Course introduction
- Computer system & programs
- Java overview, JVM, and Virtualization
- Software design principles
- Object-Oriented Programming Concepts

Computer System

- Hardware
  - The physical parts: CPU, keyboard, disks
- Software
  - System software: a set of programs that control & manage the operations of hardware, e.g., OS
  - Application software: a set of programs for end users to perform specific tasks, e.g., browser, media player

What is a program?



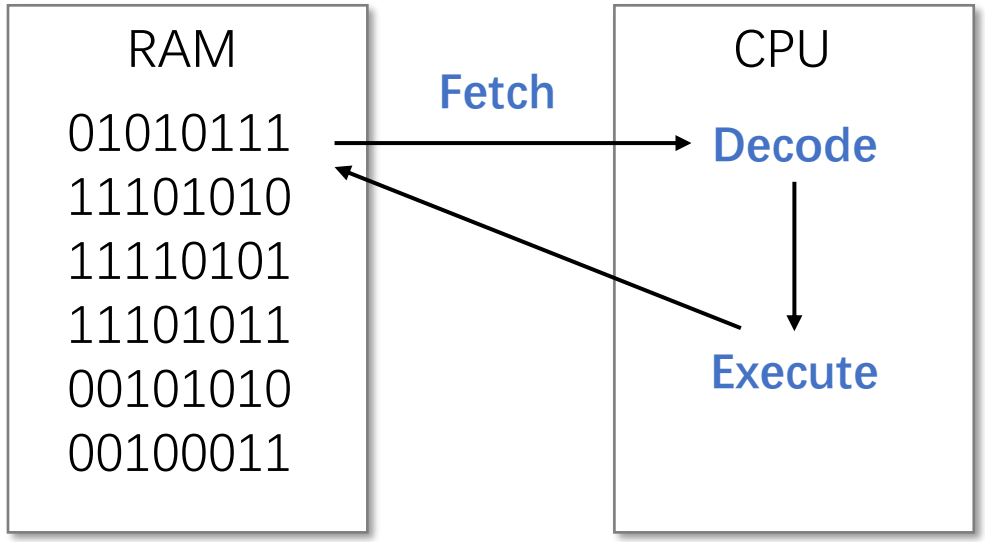
Programs

- A sequence of instructions that specifies how to perform a computation

Fetch-Decode-Execute Cycle

- **Fetch**: Get the next instruction from memory
- **Decode**: Interpret the instruction
- **Execute**: Pass the decoded info as a sequence of control signals to relevant CPU units to perform the action

The fetch-execute cycle was first proposed by **John von Neumann**, who is famous for the **Von Neumann architecture**, which is being followed by most computers today



Programs

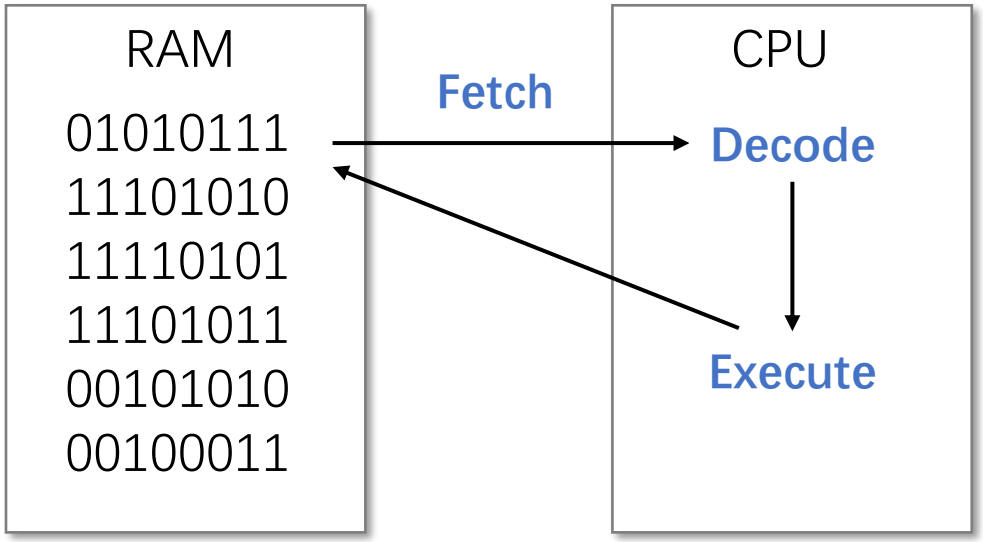
- A sequence of instructions that specifies how to perform a computation

Machine-language instructions are hard to read & write for human.

```
8B542408 83FA0077 06B80000 0000C383
FA027706 B8010000 00C353BB 01000000
B9010000 008D0419 83FA0376 078BD989
C14AEBF1 5BC3
```

A function in hexadecimal (十六进制) to calculate Fibonacci number

Source: https://en.wikipedia.org/wiki/Low-level_programming_language



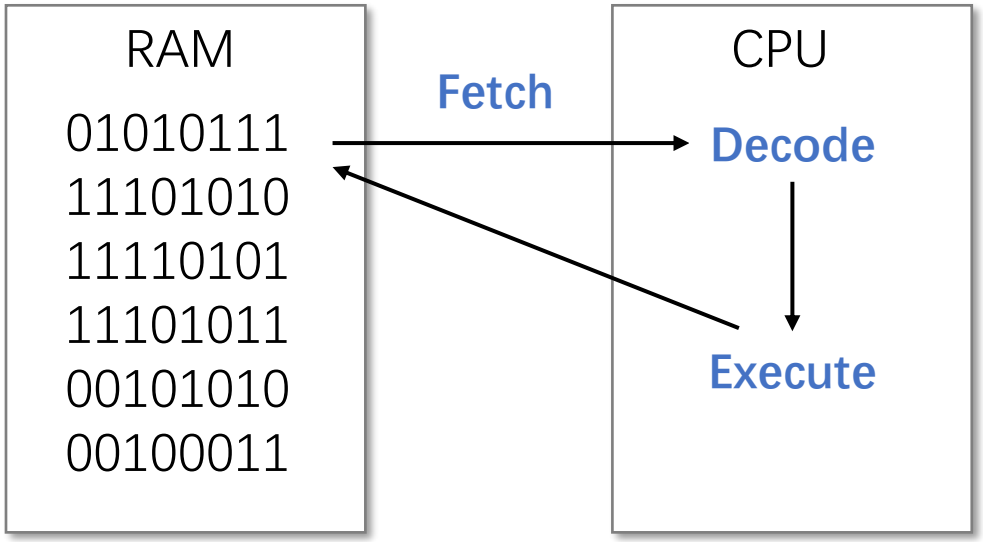
Programs

- A sequence of instructions that specifies how to perform a computation

Low-level language provides a level of abstraction on top of machine code

```
_fib:
        movl $1, %eax
        xorl %ebx, %ebx
.fib_loop:
        cmpl $1, %edi
        jbe .fib_done
        movl %eax, %ecx
        addl %ebx, %eax
        movl %ecx, %ebx
        subl $1, %edi
        jmp .fib_loop
.fib_done:
        ret
```

A function in assembly (汇编) to calculate Fibonacci number



Source: https://en.wikipedia.org/wiki/Low-level_programming_language

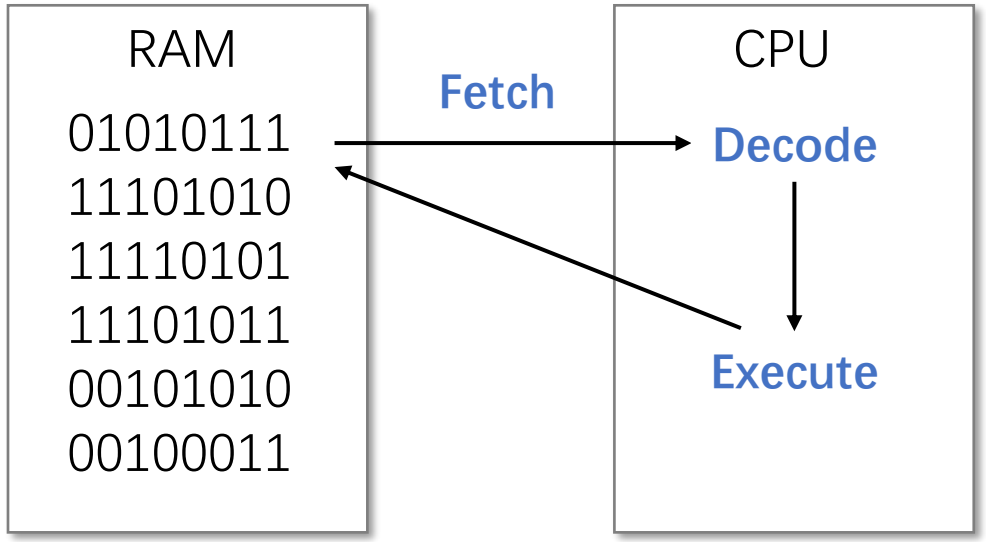
Programs

- A sequence of instructions that specifies how to perform a computation

Low-level language provides a level of abstraction on top of machine code

A video game written in assembly

Source: https://en.wikipedia.org/wiki/Prince_of_Persia_(1989_video_game)



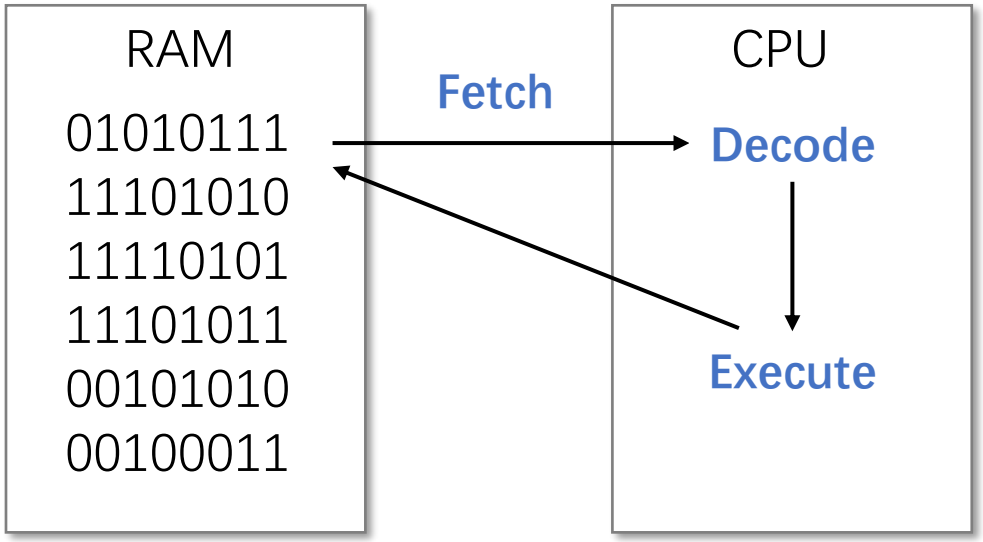
Programs

- A sequence of instructions that specifies how to perform a computation

High-level language (e.g., C++, Java, Python, etc.) provides stronger abstraction and resembles more of natural language

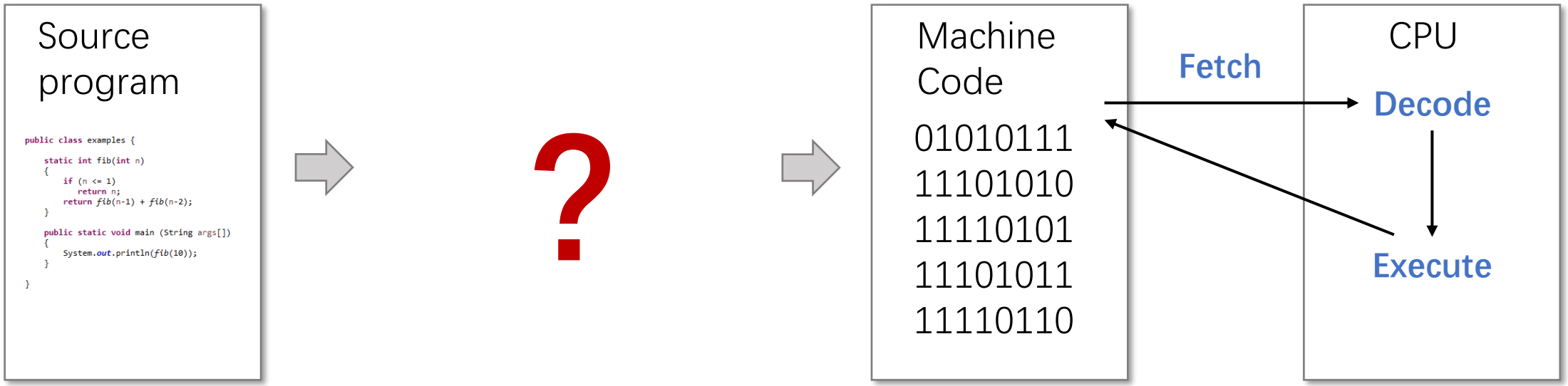
```
public class examples {

    static int fib(int n)
    {
        if (n <= 1)
           return n;
        return fib(n-1) + fib(n-2);
    }

    public static void main (String args[])
    {
        System.out.println(fib(10));
    }

}
```

A function in Java to calculate Fibonacci number

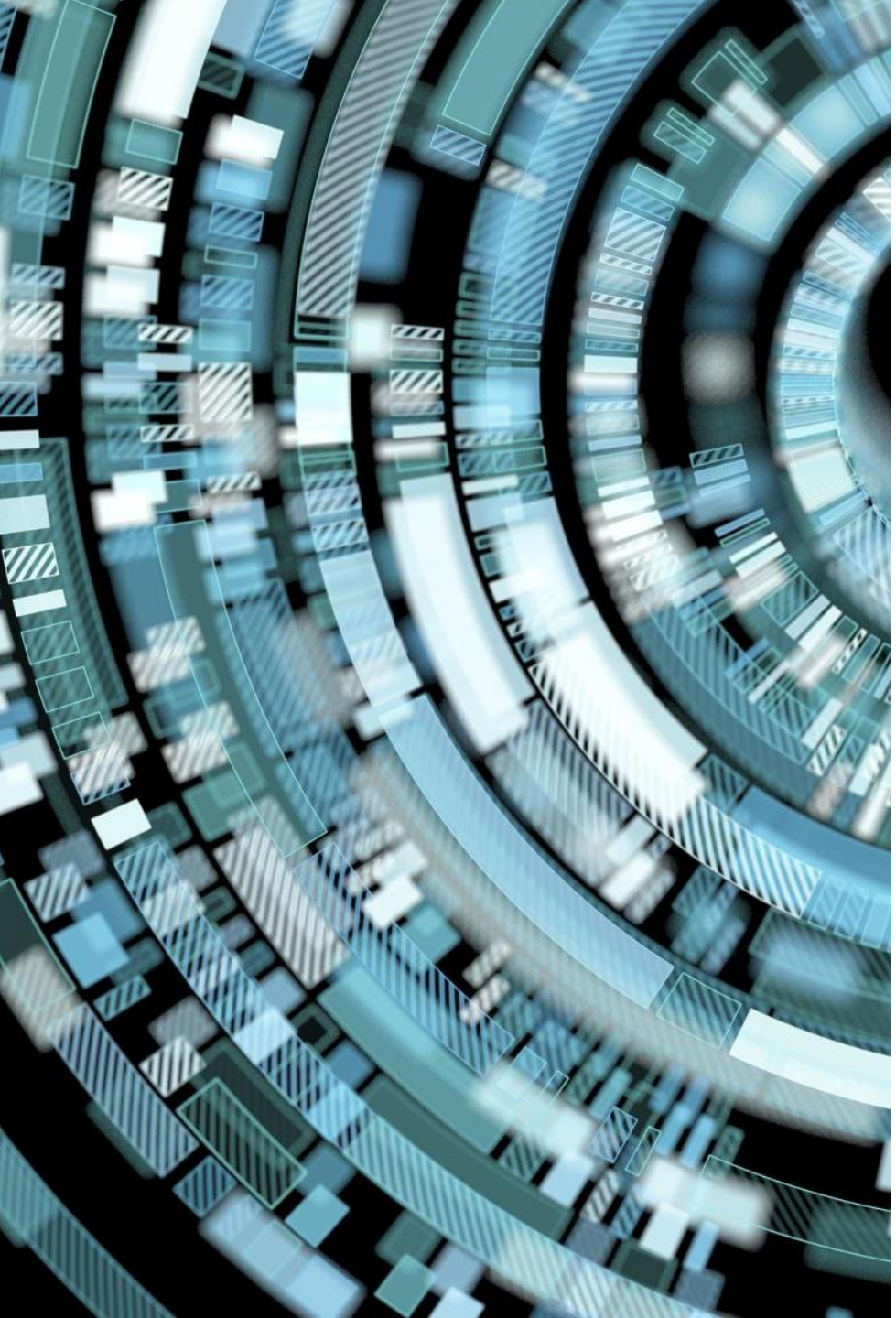


Programs

- A sequence of instructions that specifies how to perform a computation



CS202. Computer Organization

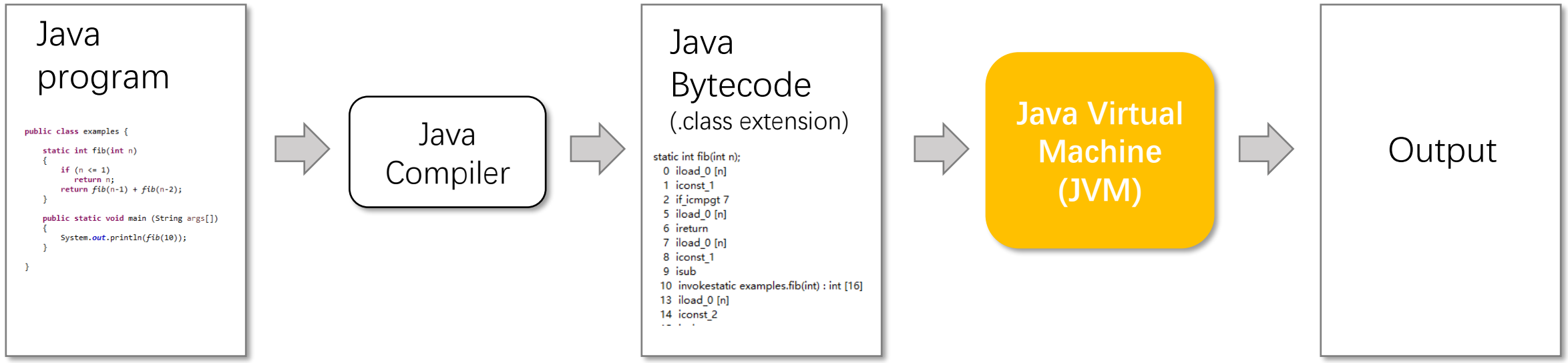


Lecture 1

- Course introduction
- Computer system & programs
- Java overview, JVM, and Virtualization
- Software design principles
- Object-Oriented Programming Concepts

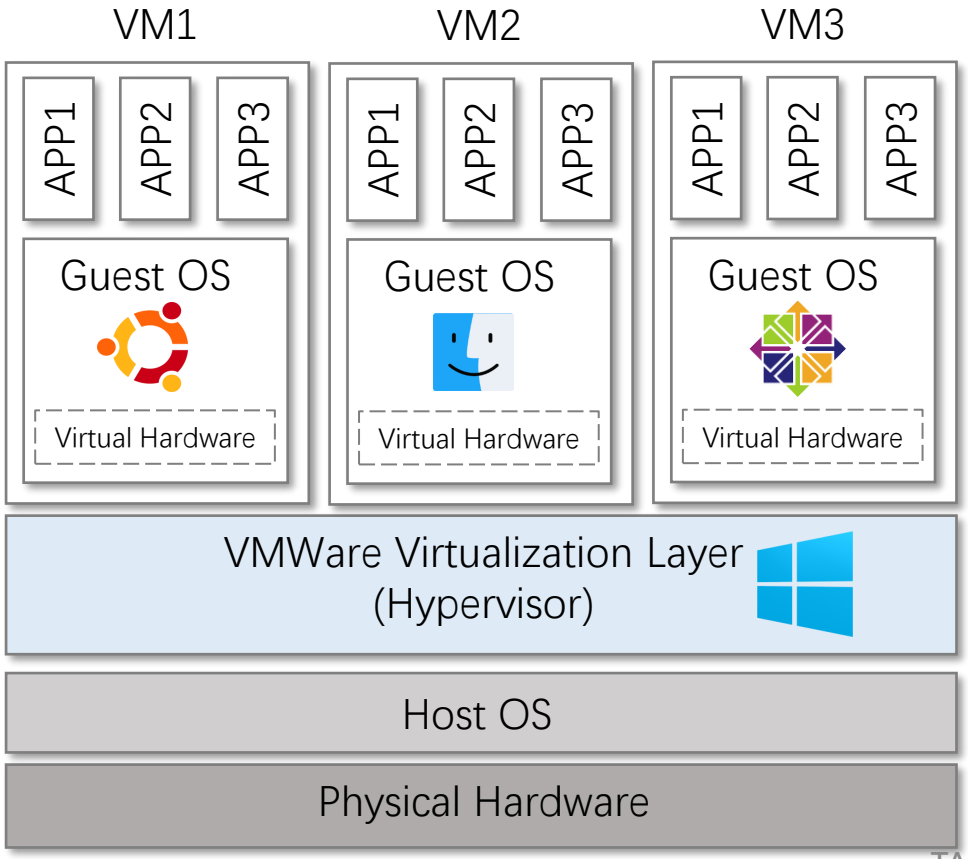
How is a Java program executed?

- Same principle: high-level source → low-level/machine code



Virtualization

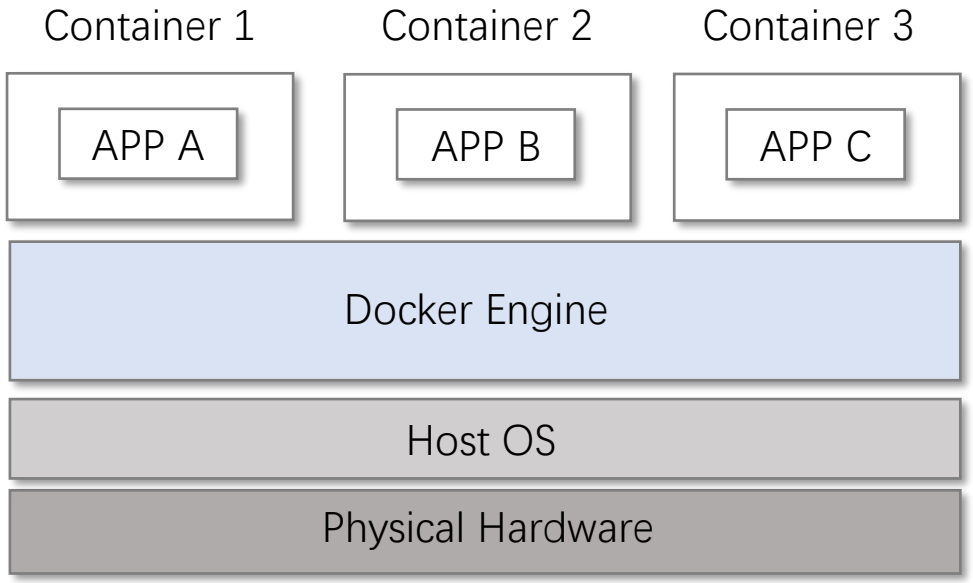
- Creating a virtual (instead of actual) version of something, such as hardware, server, operating system, etc., hiding the physical characteristics of the computing platform



- Physical computer (host machine) runs the host OS
- Virtualization uses a software layer (hypervisor) to simulate the hardware
- Different guest OS could be created, which interacts with the virtual hardware

Virtualization

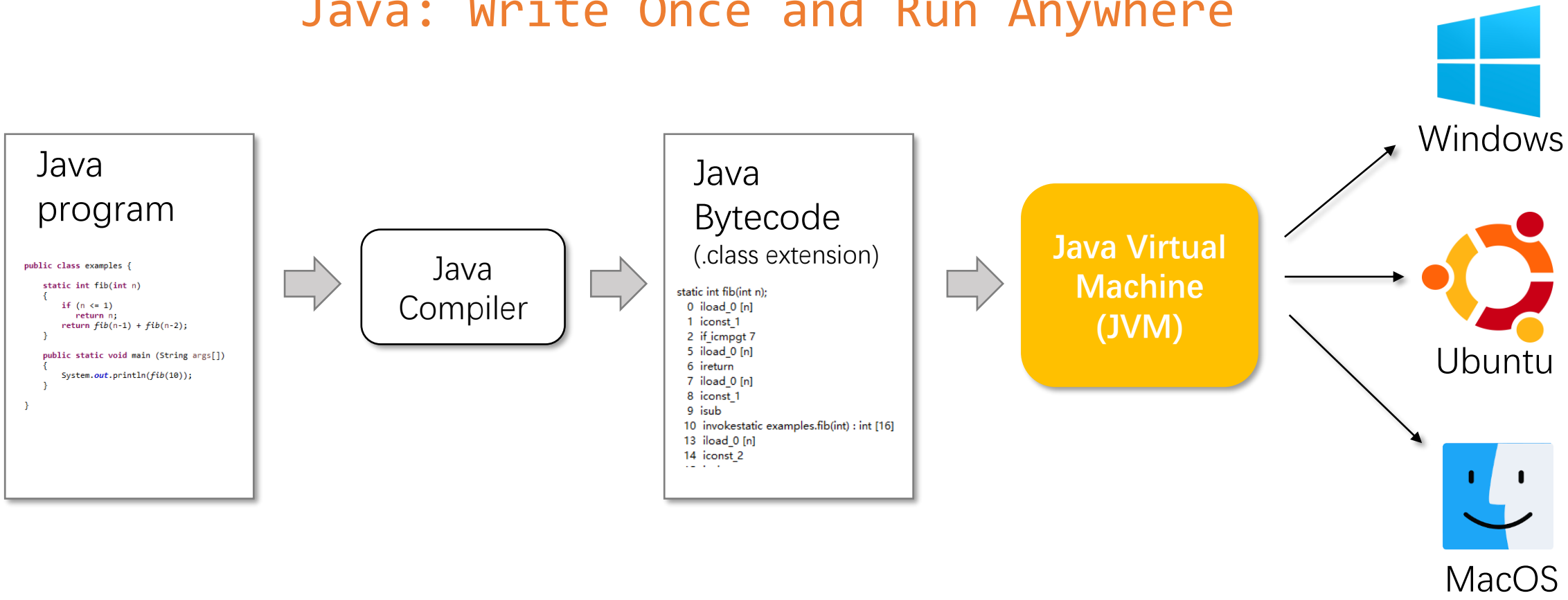
- Creating a virtual (instead of actual) version of something, such as hardware, server, operating system, etc., hiding the physical characteristics of the computing platform



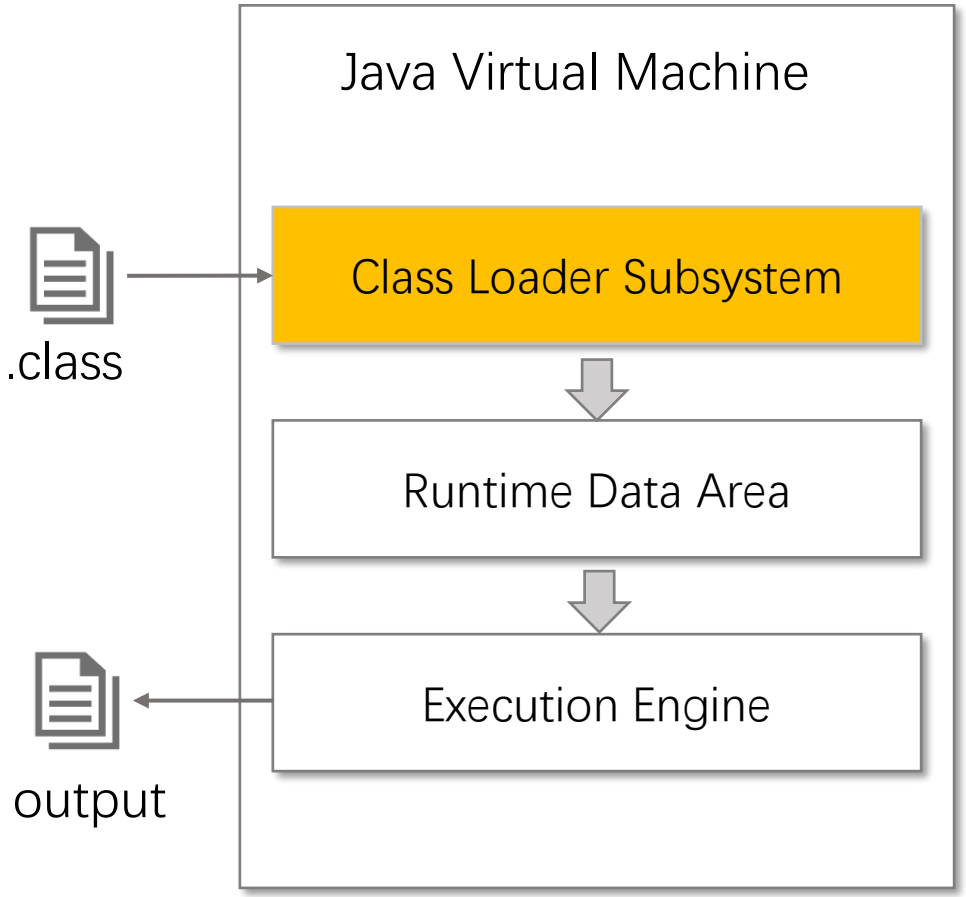
- A container consists of all the dependencies required to run an application, and isolates these dependencies from other containers on the same machine
- Containers virtualize the OS
- More lightweight, more portable

Java Virtual Machine (JVM)

Java: Write Once and Run Anywhere



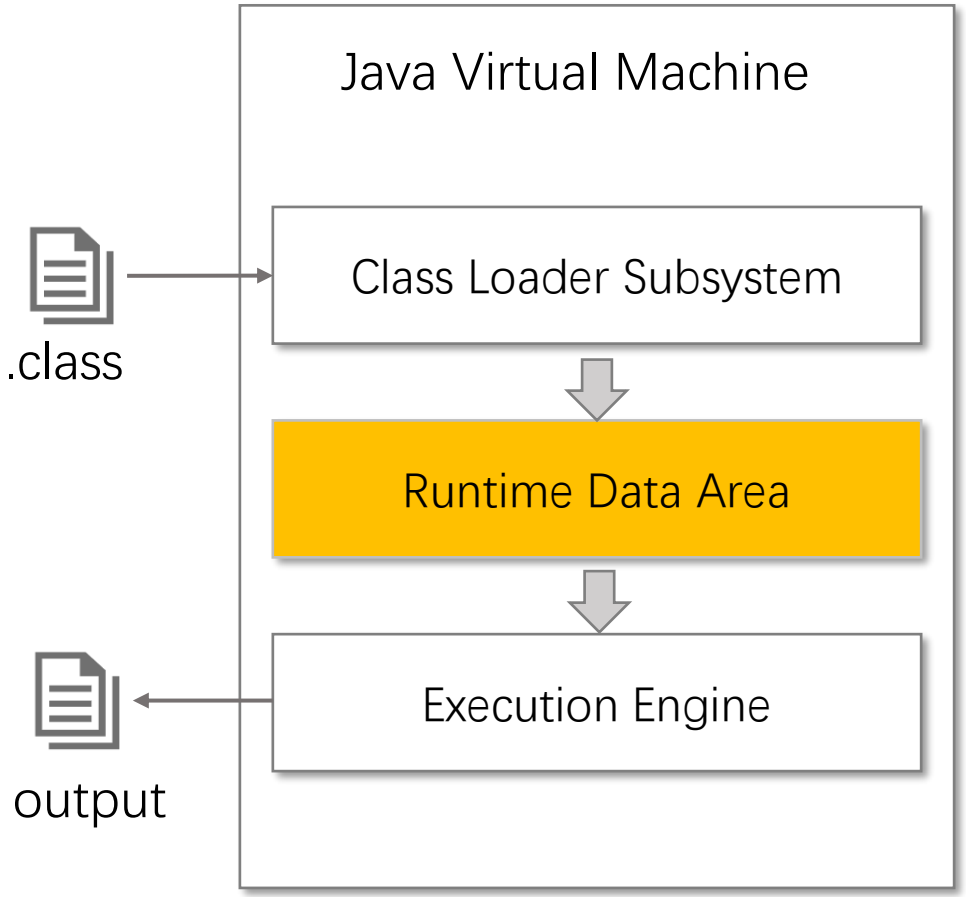
Java Virtual Machine (JVM)



Class Loader

- Locating and loading necessary .class or .jar (**J**ava **AR**chive, aggregations of .class files) files into memory
  - .jar that offers standard Java packages (e.g., java.lang, java.io)
  - .class and .jar (dependency) for your application, which is specified in *classpath*
- Errors occur when class loader fails to locate a required .class

Java Virtual Machine (JVM)

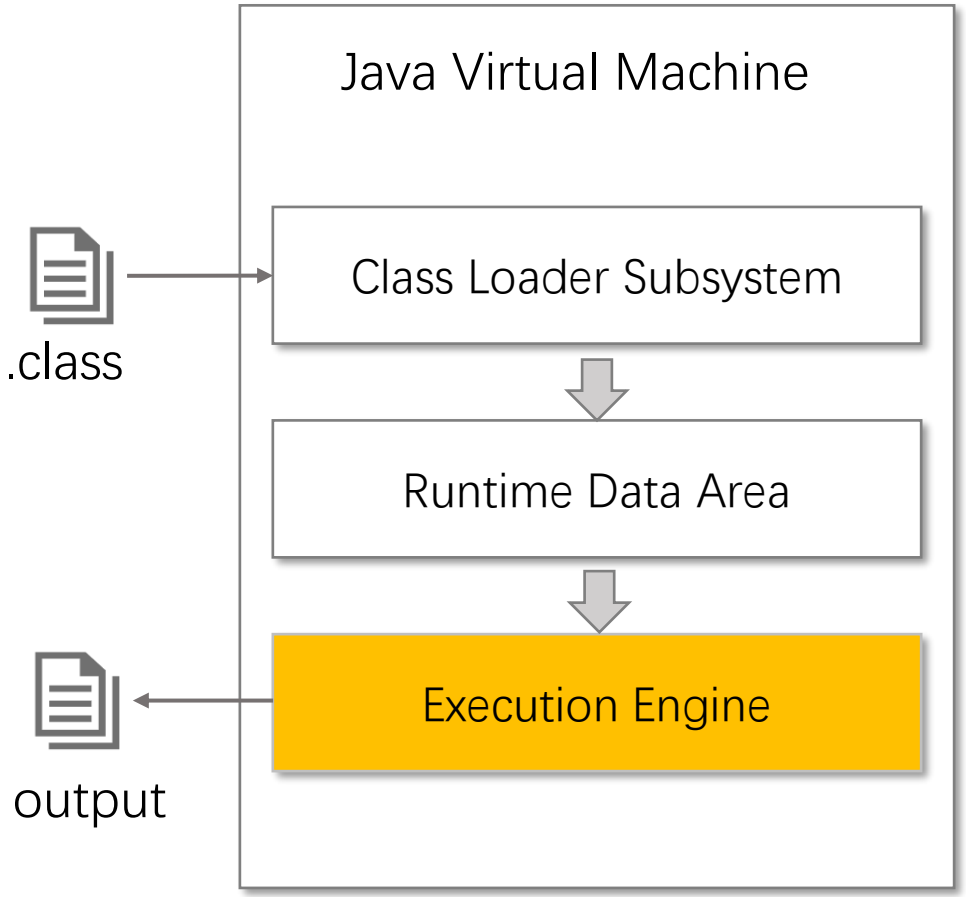


Runtime Data Area

Store all kinds of data and information

- Class-level data in Method Area
- Objects/instances in Heap Area
- Local variables in Stack Area

Java Virtual Machine (JVM)



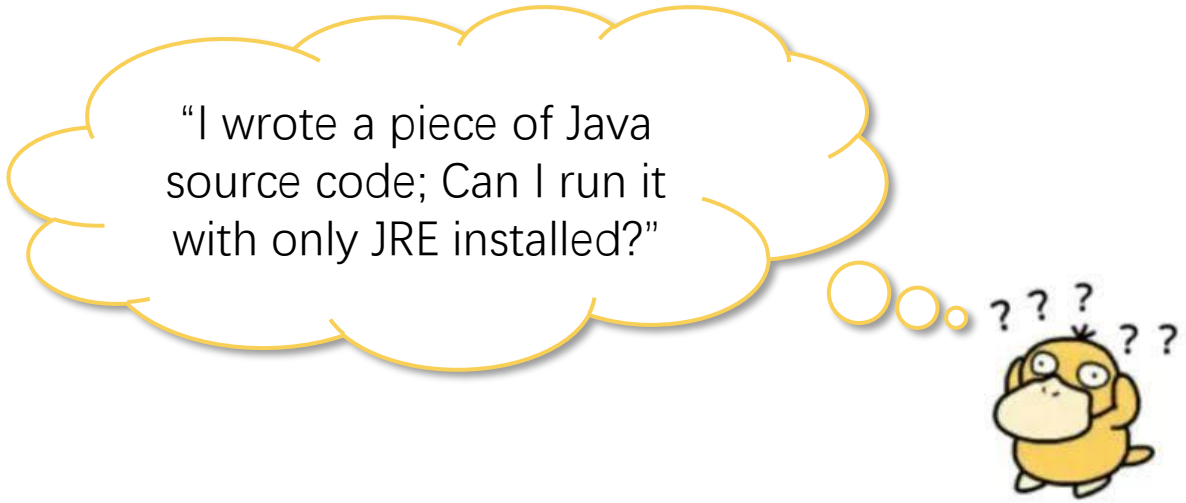
Execution Engine

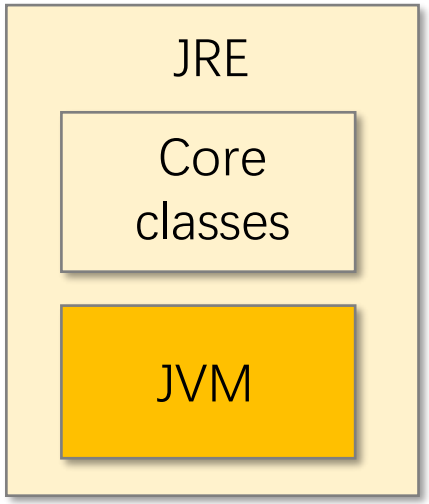
- Translating “run anywhere” .class code to “run on this particular machine” instructions
- Translation is done by Interpreter and JIT Compiler (also for optimization)
- Finally, garbage collector identifies objects that are no longer in use and reclaims the memory

JVM, JRE, and JDK

JRE: **J**ava **R**untime **E**nvironment

- Contains JVM and Core Java Classes (e.g., java.io, java.lang) for built-in functionalities
- Could be used to execute Java programs or applications

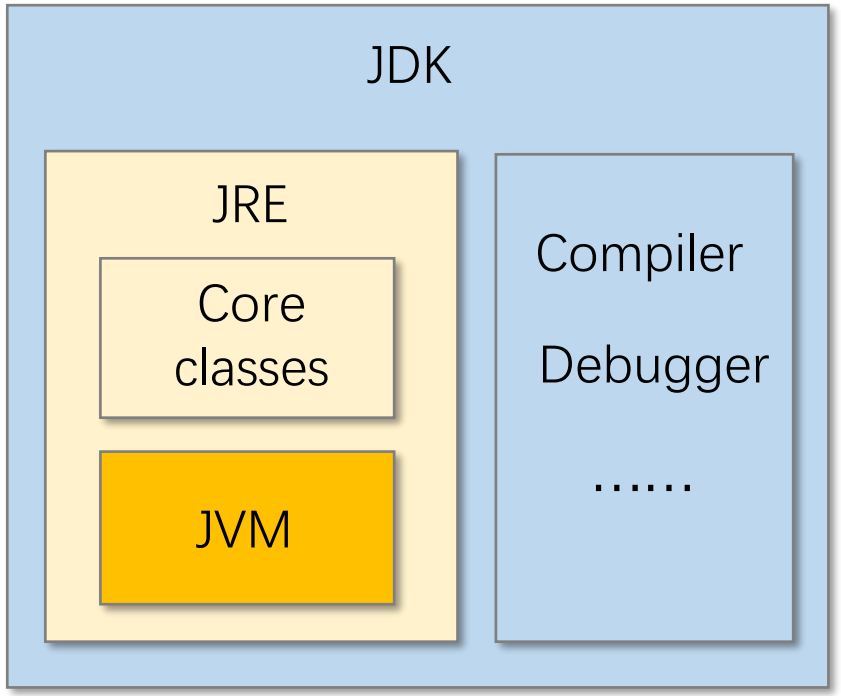


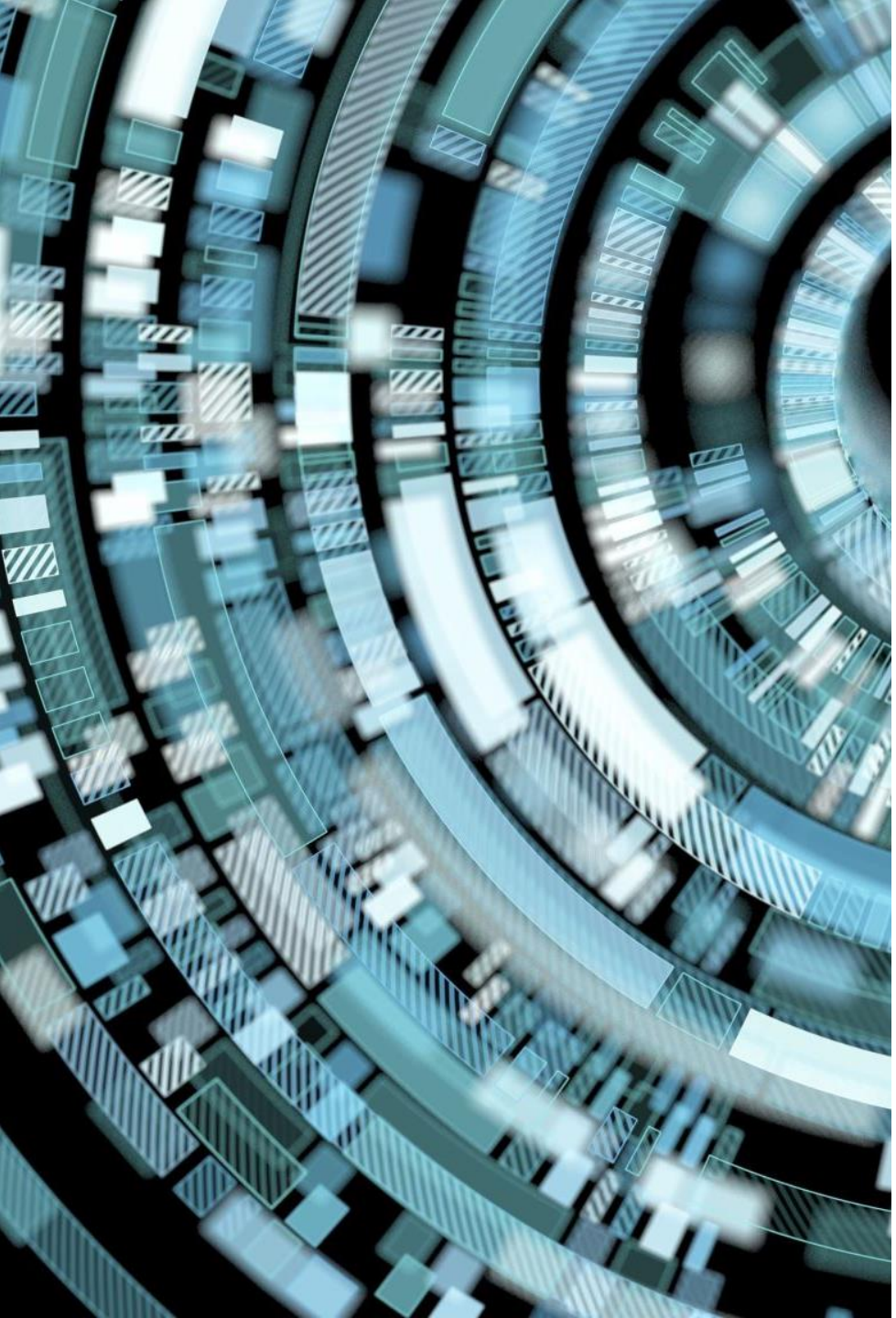


JVM, JRE, and JDK

JDK: **J**ava **D**evelopment **K**it

- Contains JRE and development tools, e.g., compiler, debugger, etc. (no need to install JRE separately if JDK is already installed)
- Compiler transform source code to byte code (.class) then JRE kicks in
- Usage scenarios for JRE and JDK





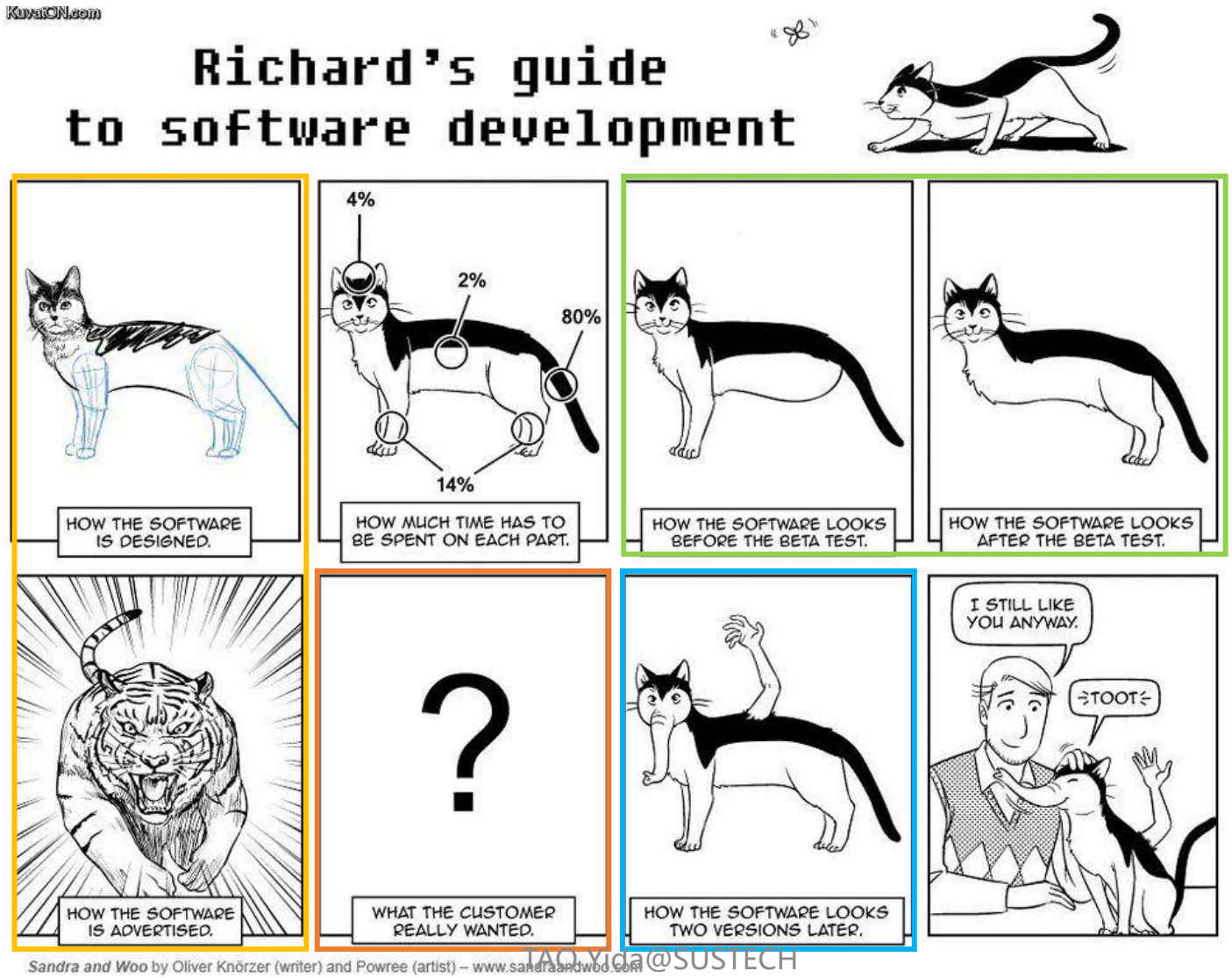
Lecture 1

- Course introduction
- Computer system & programs
- Java overview, JVM, and Virtualization
- Software Design Principles
- Object-Oriented Programming Concepts

Software design & development are complex

Requirement is evolving, sometimes deviates from the original design a lot

Requirement is hard to define, even customers themselves don't even know



Changes to one part could mysteriously affect other parts

Different designs could fulfill the same functionality; Hard to evaluate.

Tools that help

A version control system to track changes and develop collaboratively

A tool to help programmers write Java code that adheres to a coding standard

Communication is vital

- Conway's Law: Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.

Enjoy the teamwork in group projects!

Software Design Principles

- High Cohesion (高内聚)
- Low Coupling (低耦合)
- Information Hiding (信息隐藏)

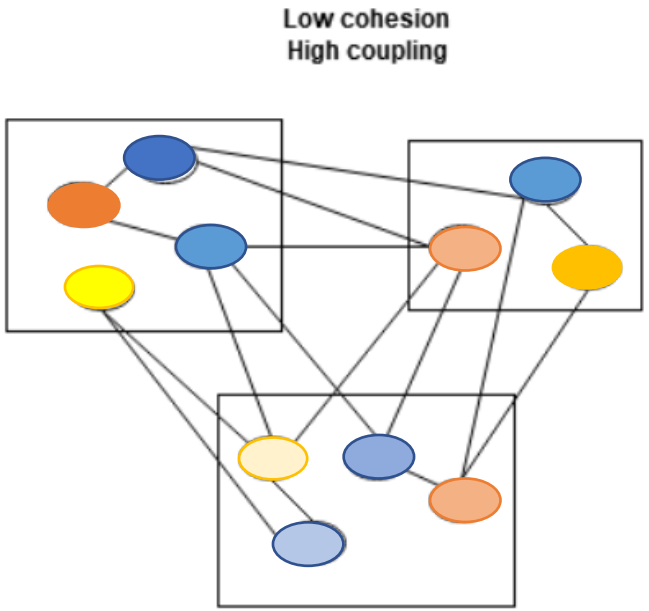
High Cohesion, Low Coupling

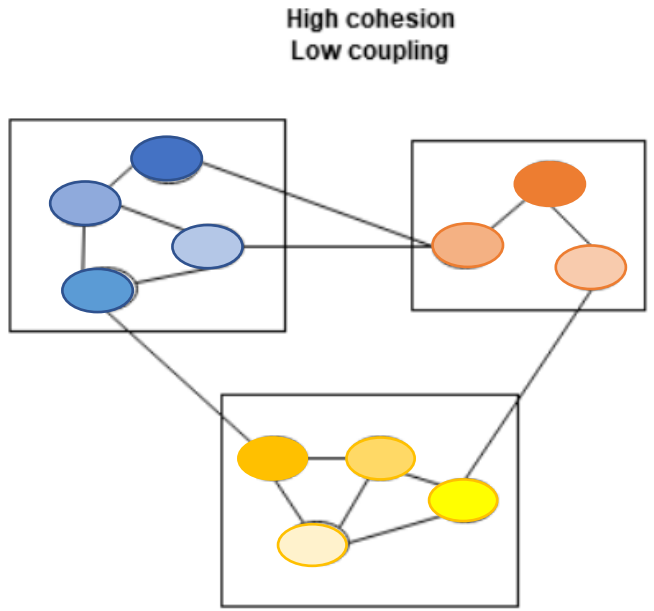
- Modules (模块): A complex software system can be divided into simpler pieces called *modules*
- Cohesion (内聚): How elements of a module are functionally related to each other
- Coupling (耦合): How different modules depend on each other

High Cohesion, Low Coupling

- High cohesion: modules are self-contained and have a single, well-defined purpose; all of its elements are directly related to the functionality that is meant to be provided by the module
- Low coupling: modules should be as independent as possible from other modules, so that changes to one module will have minimal impact on other modules

Difficult to read, understand, reuse, test, and maintain





Easy to understand, extend, and modify

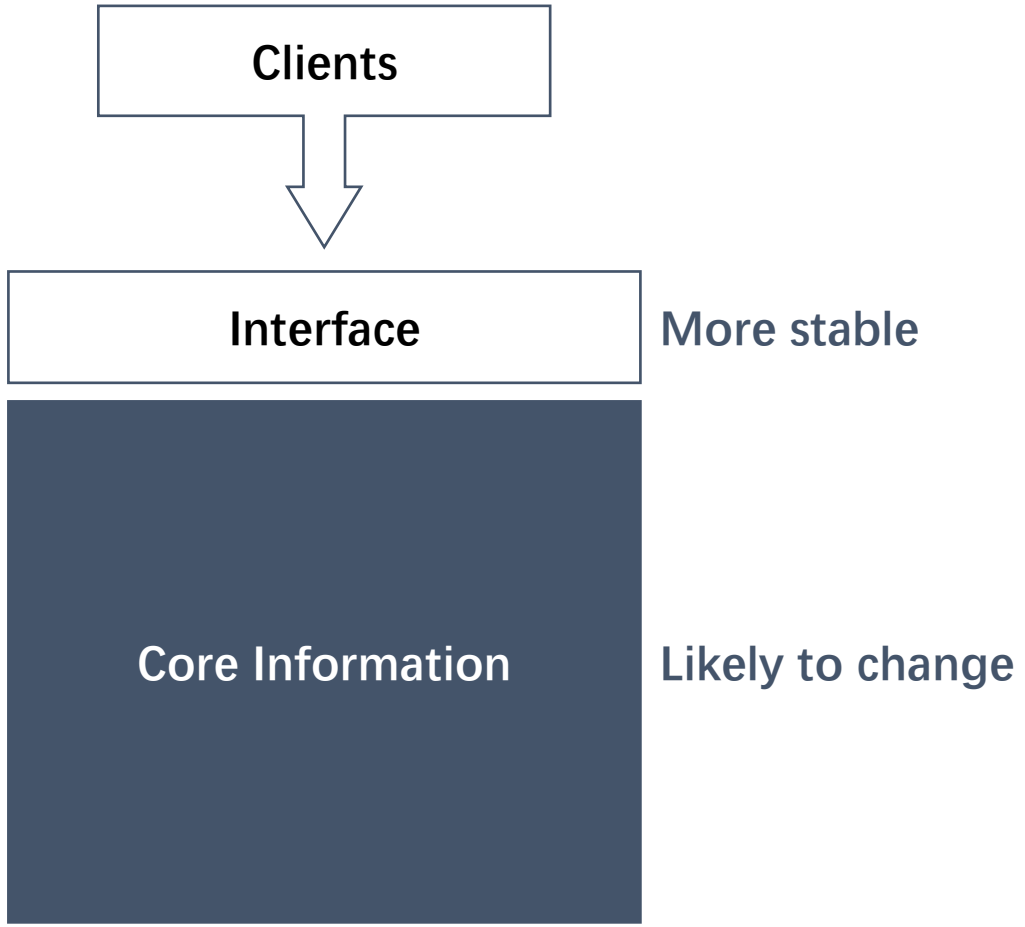
Source: Software Architecture with C++ by Adrian Ostrowski, Piotr Gaczkowski

Information Hiding

- Key idea: Hiding certain information, such as design decisions, data, and implementation details, from client programs

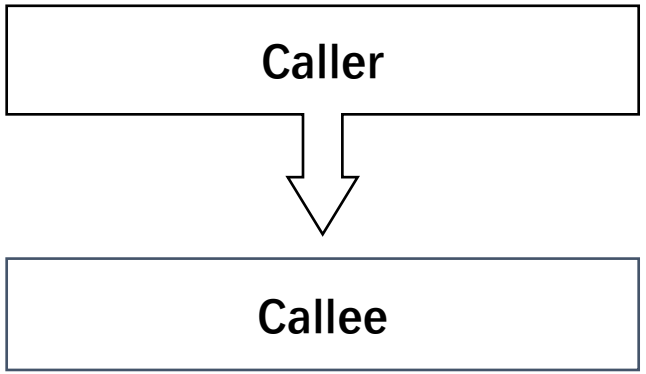
- Advantages: Client programs won't have to change even if the core design or implementation is changed

Increasing coupling -> breaking information hiding



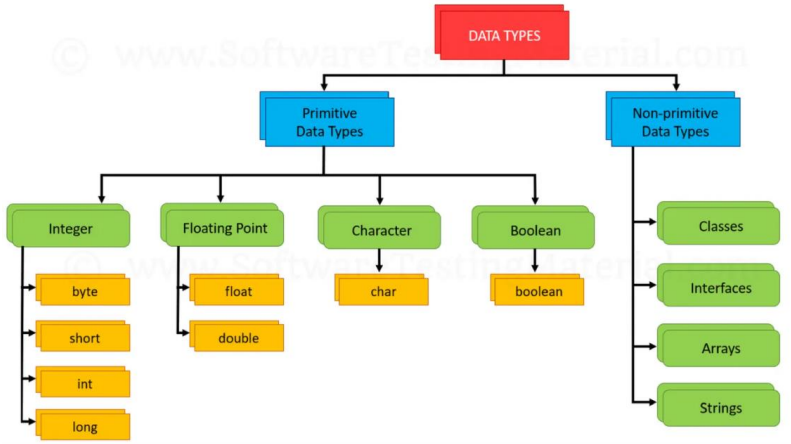
Information Hiding

Example 1. Function Call



The caller function doesn't have to know how the callee function works internally; it only has to know callee's arguments and return type

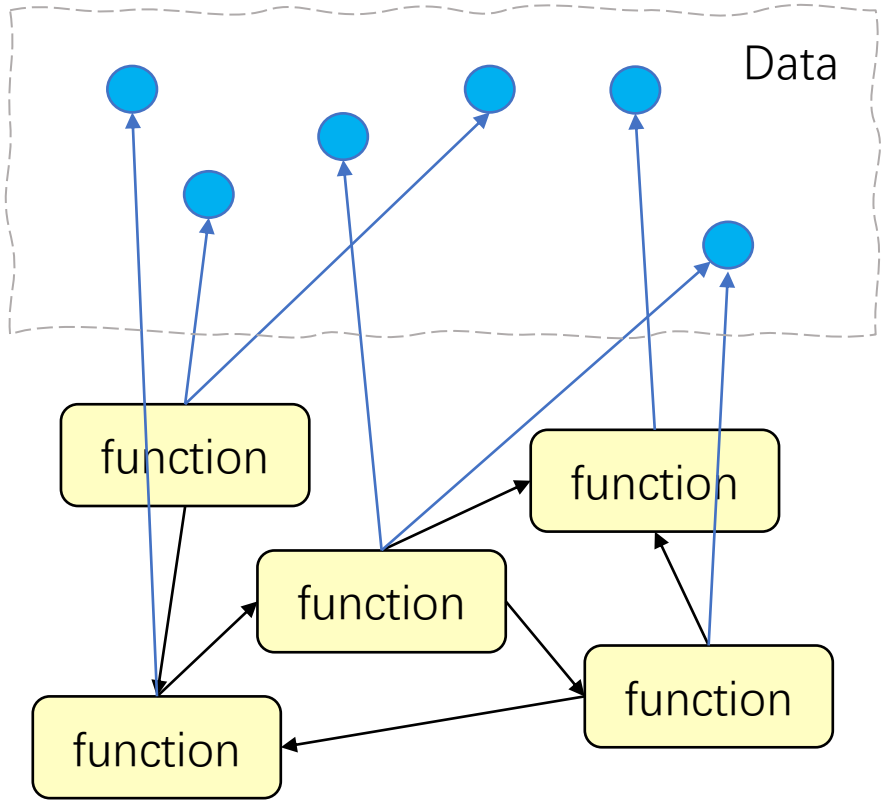
Example 2. Data Representation



You don't need to know how a data type is implemented in order to use it;

Image source: https://www.softwaretestingmaterial.com/data-types-in-java/

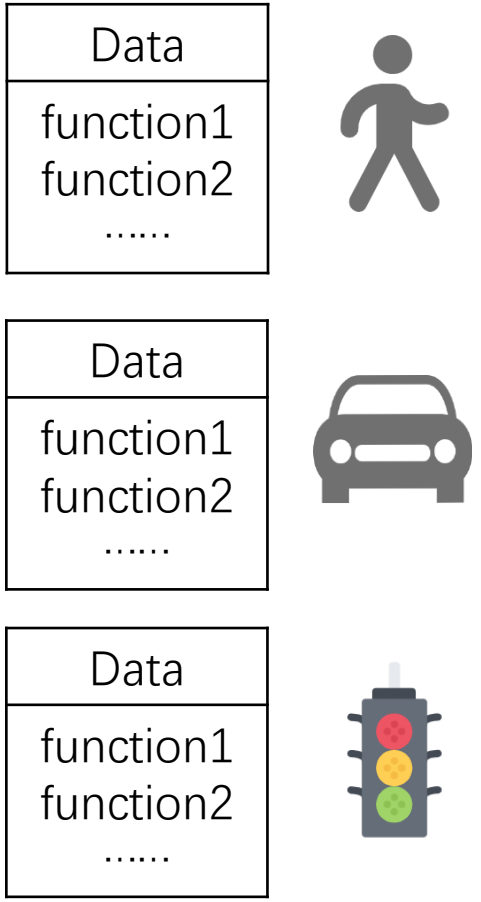
Procedural Design



High coupling. Reduced information hiding.
Hard to make changes and to scale.

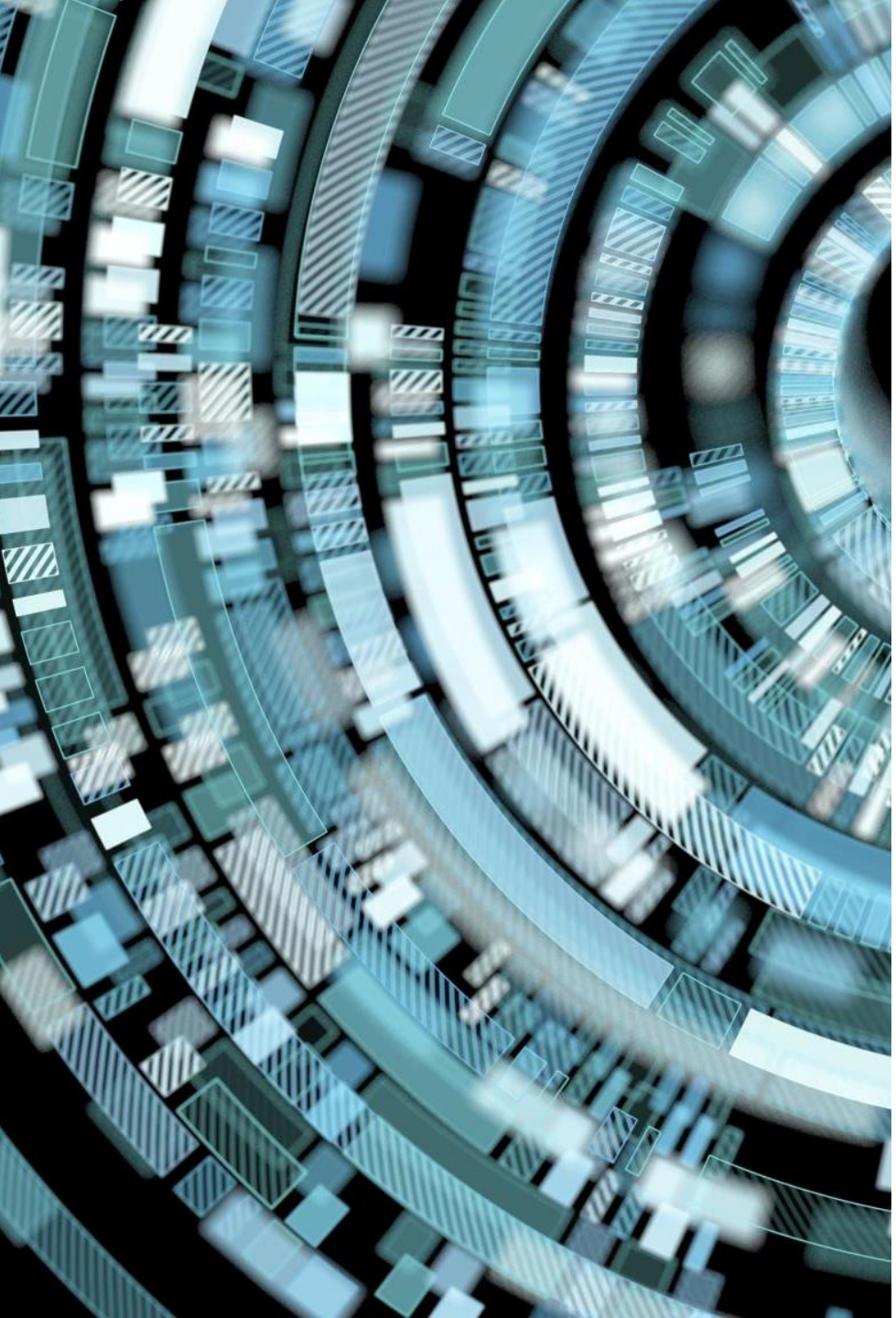
Traffic Control System

Object-oriented Design



High cohesion. Good information hiding.
Easier to maintain and extend.

Image source: https://www.hseblog.com/cross-road-safely/



Lecture 1

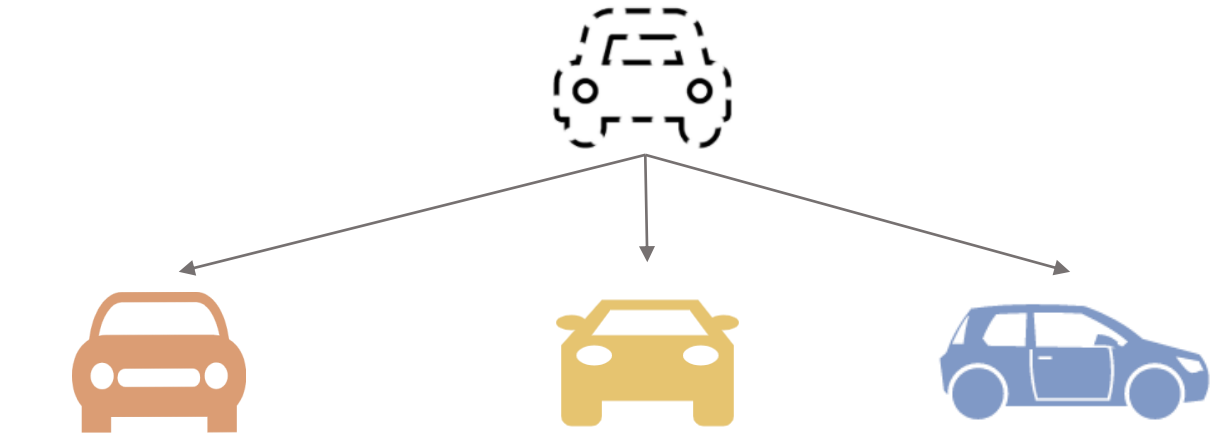
- Course introduction
- Computer system & programs
- Java overview, JVM, and Virtualization
- Software design principles
- Object-Oriented Programming Concepts

Class, Object, and Instance

- Object: Conceptually similar to real-world objects; Consist of state and behaviors. E.g., Cars have state (speed, color, model) and behavior (move, turn, stop).
- Class: a template or blueprint that is used to create objects. Consist of fields (hold the states) and methods (represent the behaviors)
  - A given object is an instance of a class.
  - Reference (non-primitive) data type.

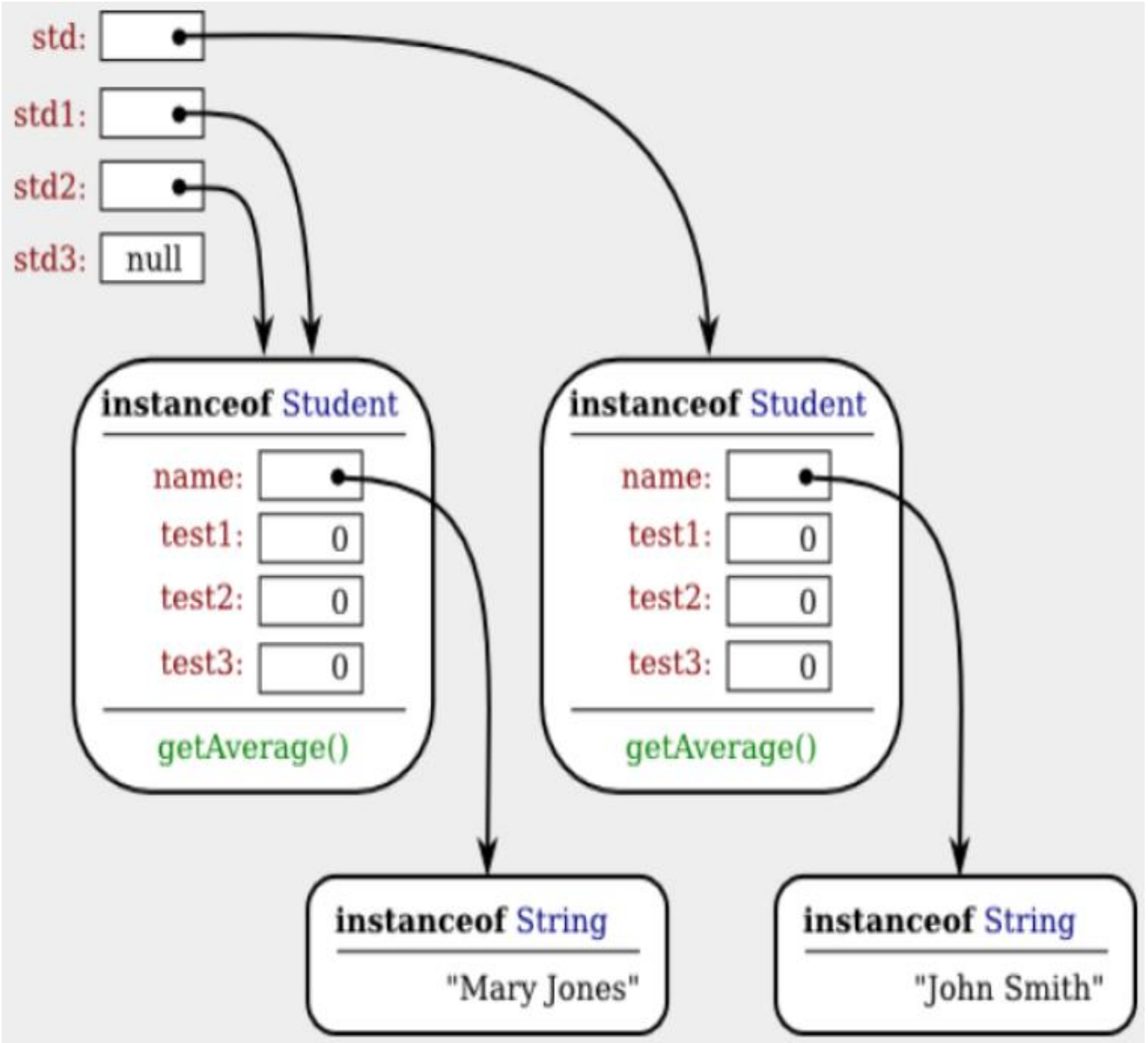
Car Class

Color Size Model
Start() Stop() Move() Turn()



```
public class Student {

   public String name;  // Student's name.
   public double test1, test2, test3;   // Grades on three tests.

   public double getAverage() {  // compute average test grade
      return (test1 + test2 + test3) / 3;
   }

}  // end of class Student
```

```
Student std, std1,        // Declare four variables of
          std2, std3;     //   type Student.

std = new Student();      // Create a new object belonging
                          //   to the class Student, and
                          //   store a reference to that
                          //   object in the variable std.

std1 = new Student();     // Create a second Student object
                          //   and store a reference to
                          //   it in the variable std1.

std2 = std1;              // Copy the reference value in std1
                          //   into the variable std2.

std3 = null;              // Store a null reference in the
                          //   variable std3.

std.name = "John Smith";  // Set values of some instance variables.
std1.name = "Mary Jones";

     // (Other instance variables have default
     //    initial values of zero.)
```



How std, std1, std2, std3 are stored?

OOP basic concepts

- Encapsulation (封装)
- Abstraction (抽象)
- Inheritance (继承)
- Polymorphism (多态)

Encapsulation

- Bundling the data and functions which operate on that data into a single unit, e.g., a class in Java.
- Think of it as a protective shield that prevents the data from being accessed by the code outside this shield.

Sound familiar?

Encapsulation or information hiding is achieved by the **Access Control** mechanism in Java

Access Control

- Use access modifiers to determine whether other classes can use a particular field or invoke a particular method

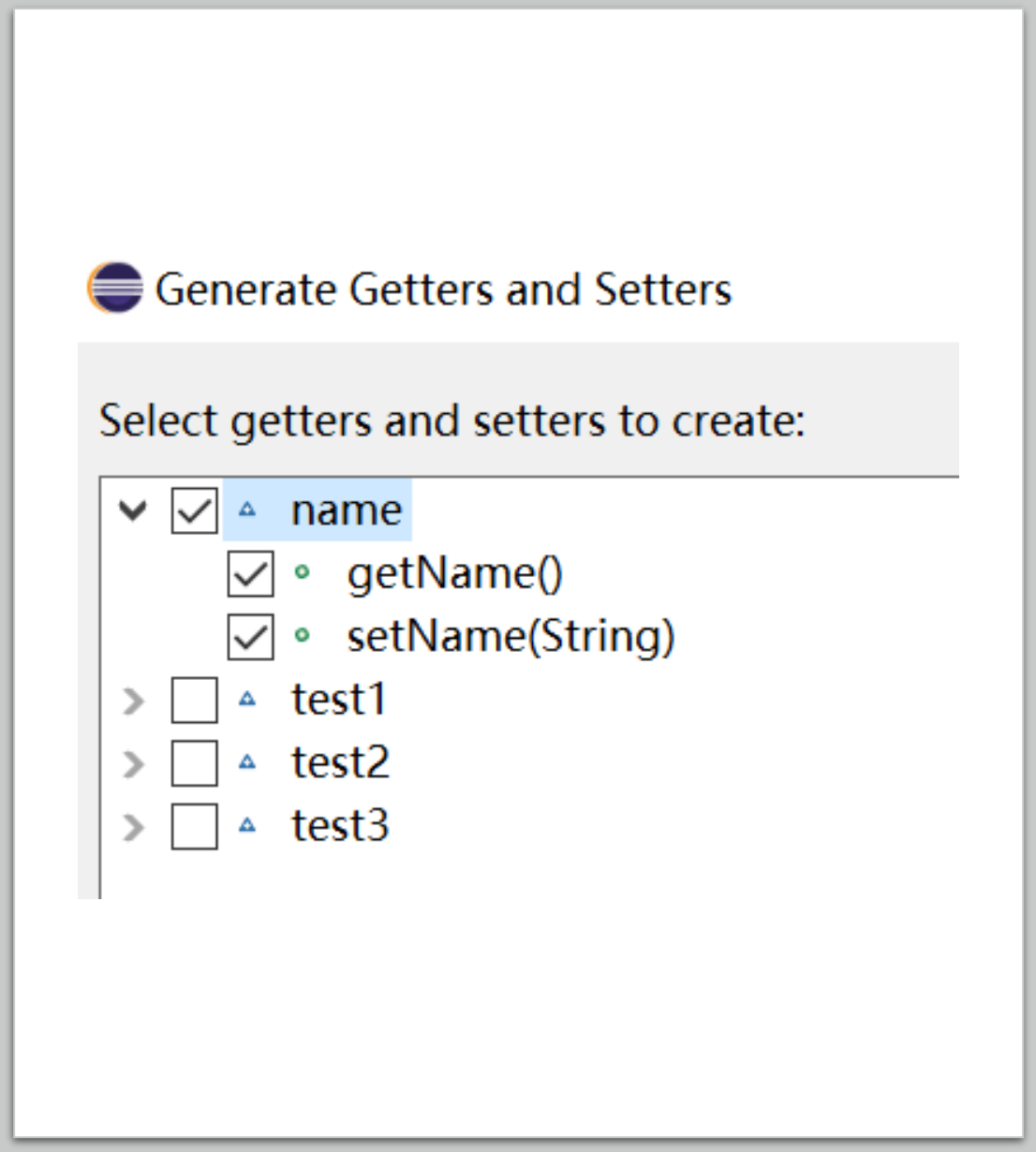
Visibility

- At the top level (class or interfaces)
  - package-private (default): visible only within its own package
  - public: visible to all classes everywhere
- At the member level (fields or methods)
  - private: can only be accessed in its own class
  - package-private (default): visible only within its own package
  - protected: can be accessed within its own package and by a subclass of its class in another package.
  - public: visible to all classes everywhere

Access Control

- Rule of thumb: always make classes or members as inaccessible as possible (using the most restricted access modifier)

- Getter and Setter
  - Getter (accessor): use getXXX() to read the data
  - Setter (mutator): use setXXX() to modify the data



Getters and Setters

```
public class Student {
        public String name;
        public double test;
}
```

```
Student std = new Student();
std.test = -1;
std.test = 200;
std.name = null;
```

Works, but makes no sense

```
public class Student {
    private String name;
    private double test;

    public void setTest(double test) {
        if(test<0 || test>100) {
        throw new IllegalArgumentException
                        ("invalid test score!");
        }
        this.test = test;
    }
}
```

```
Student std = new Student();
std.setTest(-1);
```

Getters and setters allow additional logics such as validation and error handling to be added more easily without affecting the clients

Getters and Setters

```
public class Student {
        private int[] scores = new int[]{100,90,95};

        public int[] getScores() {
                return scores;
        }
}
Student std = new Student();

int[] scores = std.getScores();
// [100, 90, 95], expected
System.out.println(Arrays.toString(scores));

scores[0] = 10;

// [10, 90, 95], Why scores, which is private, could still be modified?
System.out.println(Arrays.toString(std.getScores()));
```

Any problems with the code?

The getter method returns a reference of the internal variable scores directly, so the outside code can obtain this reference and makes change to the internal object.

Getters and Setters …?

Further Reading

- Getter Eradicator by Martin Fowler. https://martinfowler.com/bliki/GetterEradicator.html
- Tell-Don't-Ask by Martin Fowler. https://martinfowler.com/bliki/TellDontAsk.html
- Why use getters and setters? https://stackoverflow.com/questions/1568091/

OOP basic concepts

- Encapsulation (封装)
- Abstraction (抽象)
- Inheritance (继承)
- Polymorphism (多态)

Abstraction

- Identifying and providing only essential ideas to users while hiding background details

- Abstraction solves problem at design level (what should be done) while Encapsulation solves problem at implementation level (how it should be done)

- Achieved in Java by interface and abstract class

Abstract Class

- Purpose: to provide a general guideline or blueprint of a particular concept without having to implement every method; Subclasses should provide the full implementation
- Cannot be instantiated; Subclasses that *extend* the abstract class can be instantiated
- Can have concrete and abstract methods
  - Abstract methods (no implementation): Subclasses must provide the implementation
  - Concrete methods (with implementation): Subclasses could inherit or override it

```java
abstract class Shape {
    // concrete method
    void moveTo(int x, int y)
    {
        System.out.println("moved to x=" + x + " and y=" + y);
    }

    // Abstract method should be implemented by its subclass
    abstract double area();
}
```

```java
class MyRectangle extends Shape {

    int length, width;

    MyRectangle(int length, int width)
    {
        this.length = length;
        this.width = width;
    }

    @Override
    double area()
    {
        return (double)(length * width);
    }
}
```

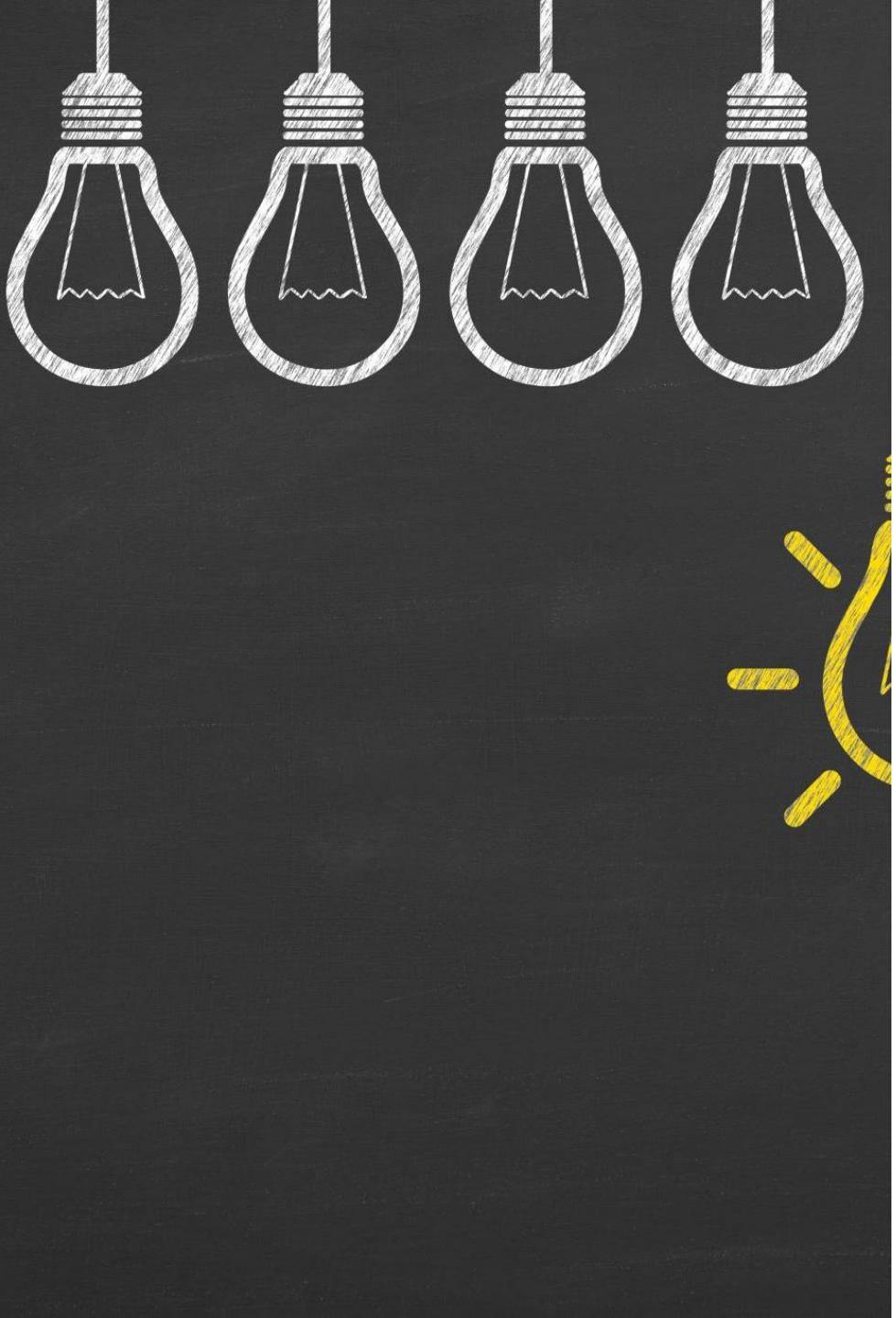
```java
class MyCircle extends Shape {

    double pi = 3.14;
    int radius;

    MyCircle(int radius)
    {
        this.radius = radius;
    }

    @Override
    double area()
    {
        return (double)((pi * radius * radius));
    }
}
```

```java
Shape rect = new MyRectangle(2, 3);
rect.moveTo(1, 2);
System.out.println("Area:" + rect.area());
```

```
moved to x=1 and y=2
Area:6.0
```

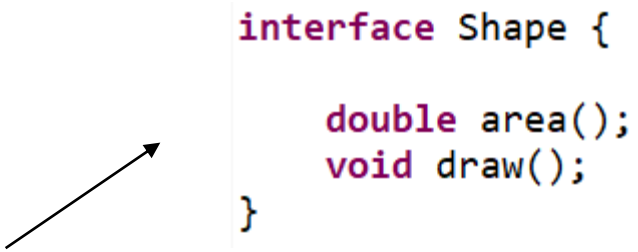
```java
Shape circle = new MyCircle(2);
circle.moveTo(2, 4);
System.out.println("Area:" + circle.area());
```

```
moved to x=2 and y=4
Area:12.56
```



Interface

- A group of related abstract methods with empty bodies (i.e., an *interface* or *contract* to the outside world)
- Classes that implement an interface must override all of its methods (should conform to the “contract” and implement all the behavior it promises to provide)
- Compared to Abstract Class
  - An interface cannot be instantiated; Classes that *implement* interfaces can be instantiated
  - A class can implement multiple interfaces, but can inherit only one abstract class



```java
interface Shape {

    double area();
    void draw();
}
```

```java
class MyRectangle implements Shape {

    int length, width;

    MyRectangle(int length, int width)
    {
        this.length = length;
        this.width = width;
    }


    @Override
    public double area()
    {
        return (double)(length * width);
    }

    @Override
    public void draw()
    {
        System.out.println("Draw a rectangle");
    }
}
```

```java
Shape rect = new MyRectangle(2, 3);
rect.draw();
System.out.println("Area:" + rect.area());
```

```
Draw a rectangle
Area:6.0
```

```java
class MyCircle implements Shape {

    double pi = 3.14;
    int radius;

    MyCircle(int radius)
    {
        this.radius = radius;
    }

    @Override
    public double area()
    {
        return (double)((pi * radius * radius));
    }

    @Override
    public void draw()
    {
        System.out.println("Draw a circle");
    }
}
```

```java
Shape circle = new MyCircle(2);
circle.draw();
System.out.println("Area:" + circle.area());
```

```
Draw a circle
Area:12.56
```

OOP basic concepts

- Encapsulation (封装)
- Abstraction (抽象)
- Inheritance (继承)
- Polymorphism (多态)

Inheritance

- Motivation: objects are similar and share common logics
- Inheritance allows a new class (subclass, child class, derived class) to be created by deriving variables and methods from an existing class (superclass, parent class, base class)
- Reduce code redundancy & support good code reuse

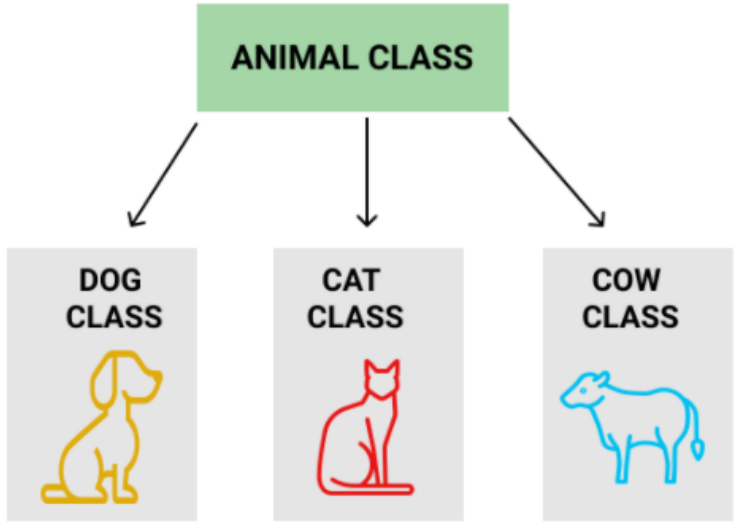
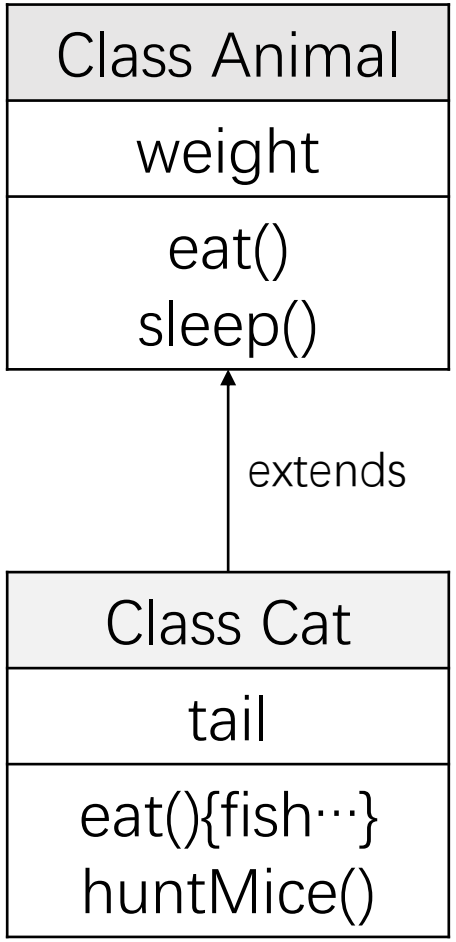


Image source: OOP Inheritance. San Joaquin Delta College. https://eng.libretexts.org/@go/page/34639

Subclass

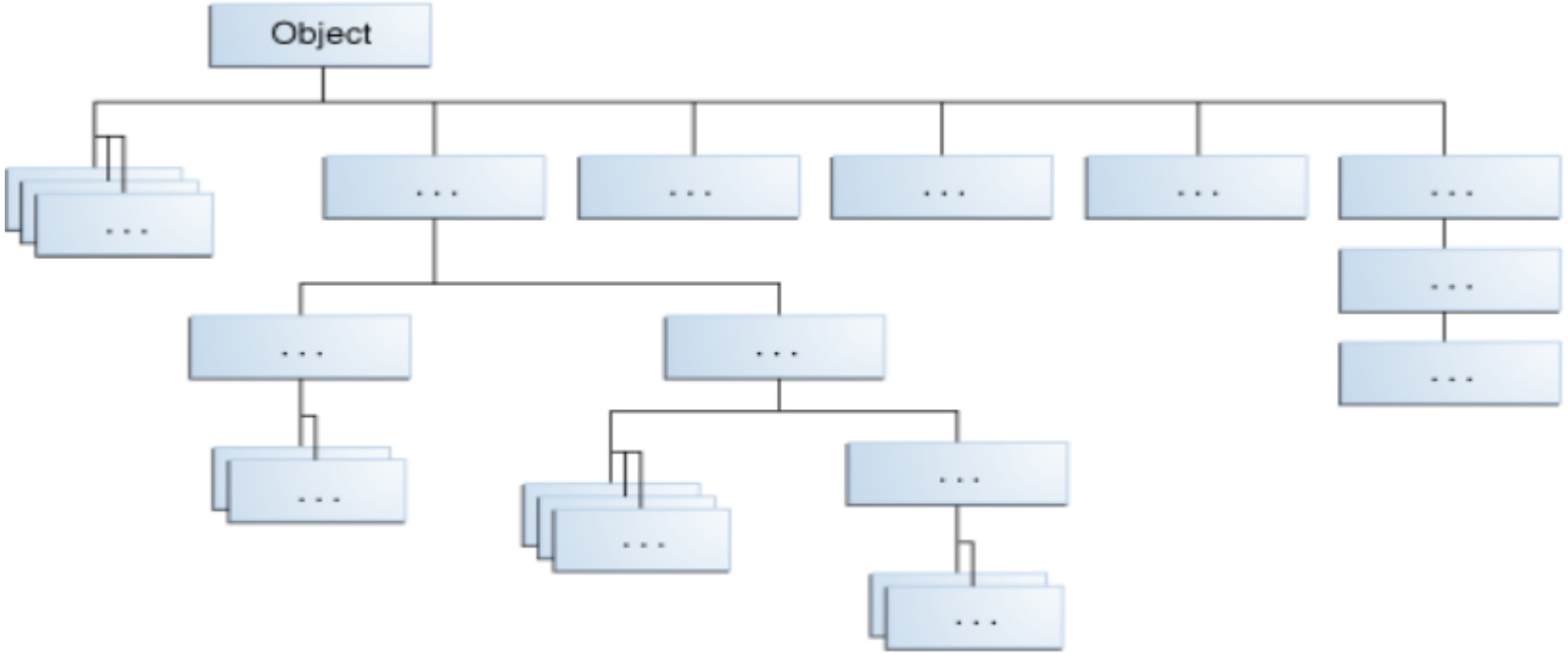
- Subclass could use inherited field directly (**`weight`**)
- Subclass could declare new fields (**`tail`**)



- Subclass could use inherited method directly (**`sleep()`**)
- Subclass could override methods in superclass (**`eat()`**)
- Subclass could declare new methods (**`huntMice()`**)

The Java Class Hierarchy

- The `Object` class (in `java.lang` package) is the parent class of all the classes



Some classes derive directly from `Object`, others derive from those classes, and so on – forming a tree-like class hierarchy

Object Class

- Providing behaviors common to all the objects, e.g., objects can be compared, cloned, notified, etc.

boolean equals(Object obj)

Indicates whether another obj is "equal to" this one; return True only if two variables refer to the same physical object in memory

```
public class Money {
    int amount;

    Money(int amount){
        this.amount = amount;
    }
}
```

false

```
@Override
public boolean equals(Object o) {
    Money other = (Money)o;
    return this.amount == other.amount;
}
```

true

```
Money m1 = new Money(100);
Money m2 = new Money(100);
boolean compare = m1.equals(m2);
```

Object Class

- Providing behaviors common to all the objects, e.g., objects can be compared, cloned, notified, etc.

`String toString()`
Returns a string representation of the object. Default is the name of the class + "@" + hashCode

```
public class Money {
    int amount;

    Money(int amount){
        this.amount = amount;
    }
}
```

```
Money m = new Money(100);
System.out.println(m);
```

```
@Override
public String toString() {
    return "Amount is " + amount;
}
```

```
Money@515f550a      Amount is 100
```

OOP basic concepts

- Encapsulation (封装)
- Abstraction (抽象)
- Inheritance (继承)
- Polymorphism (多态)

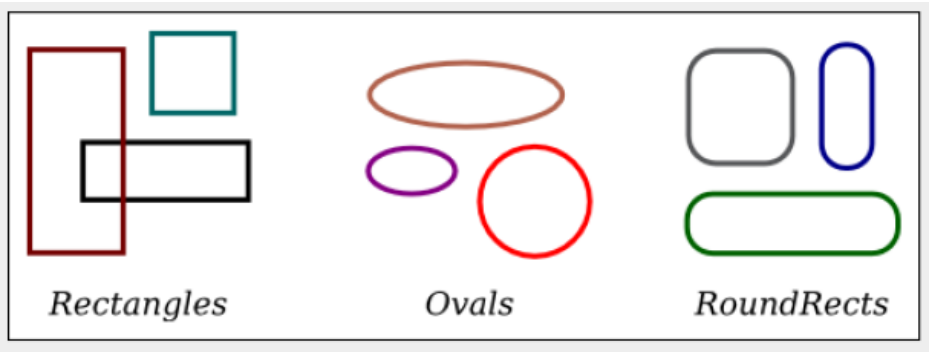
Polymorphism

- An object could take many forms
- The same action could be performed in many different ways

- Suppose that `shapelist` is a variable of type `Shape[]`; the array has already been created and filled with data.
- Some of the elements in the array are `Rectangles`, some are `Ovals`, and some are `RoundRects`
- Implementations for drawing are different, but we don't have to declare different `draw()`

```
for (int i = 0; i < shapelist.length; i++ ) {
    Shape shape = shapelist[i];
    shape.redraw();
}
```

Same action



Many forms

Binding

- Mapping the name of the method to the final implementation.
- Static binding vs Dynamic binding

Static binding (early binding)

- Mapping is resolved at compile time
- Method overloading (methods with the same name but different parameters) are resolved using static binding

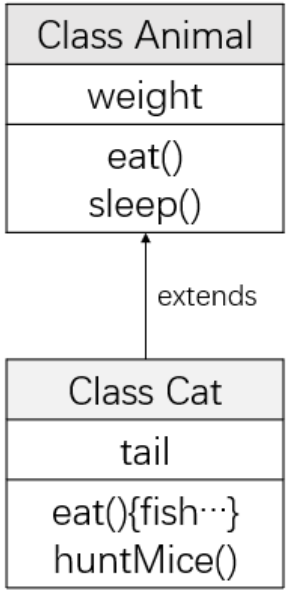
```
class Calculator{
    public int sum(int a, int b){
        return a+b;
    }

    public int sum(int a, int b, int c){
        return a+b+c;
    }
}
```

Binding

- Mapping the name of the method to the final implementation.
- Static binding vs Dynamic binding

Dynamic binding (late binding)

- Mapping is resolved at execution time
- Method overriding (subclass overrides a method in the superclass) are resolved using dynamic binding




```
Animal x = new Cat();
x.eat();
```

- ✓ Compilation ok, since Animal type has eat() method
- ✓ At execution time, x refers to a Cat object, so invoking Cat's eat() method

Next Lecture

- Generics
- ADT
- Collections