Chapter 2: Java OOP I

Xiang Zhang

javaseu@163.com

https://wdsseu.github.io/java/

Content

- OO Concepts
- Class and Objects
  - Package
  - Field
  - Method
  - Main method
  - Object
  - Construct and Initialization
  - Access Control

Object Oriented Programming

- Object

- Class

- Abstraction
  - Design / Implementation

- Inheritance
  - Common / Special

- Polymorphism
  - Method / Behavior

- Example:

  People live in houses(Class), and houses usually look like this(abstraction). I have my own house (Object). It is a little special because it is in Georgian style (Inheritance). So you can call it a house or a Georgian house.


```
House zhang_house = new GeorgianHouse();
```

- Class: concepts that have attributes and behavior

```java
public class Person {
    private String name;
    private int age;
```

- Object: instances that can interact

```java
public static void main(String[] args) {
    Person tom = new Person();
    tom.greet();
```

- Abstraction: design first, implement later

```java
public abstract class AbstractPerson {
    public void greet() {}
}
```

- Inheritance: reuse the code by parent-child classes

```java
public class Student extends Person{
    public void study(Course course) {...}
}
```

- Polymorphism: same method, different behavior

```java
Bird p = new Parrot();
p.fly(SPL, JLH);
p.tweet();
```

Example on Polymorphism

```java
public class Bird{
    public void tweet(){System.out.println("Jiu~Jiu~!");}
    public void fly(Place a, Place b){...}
}

public class Parrot extends Bird{
    public void tweet(){System.out.println("Hello!");}
    public void eat() {...}
}

......
```

declared type

runtime type / actual type

```java
Bird p = new Parrot();
p.fly(JLH, SPL);
p.tweet();
```

is it correct?

```java
p.eat();
```
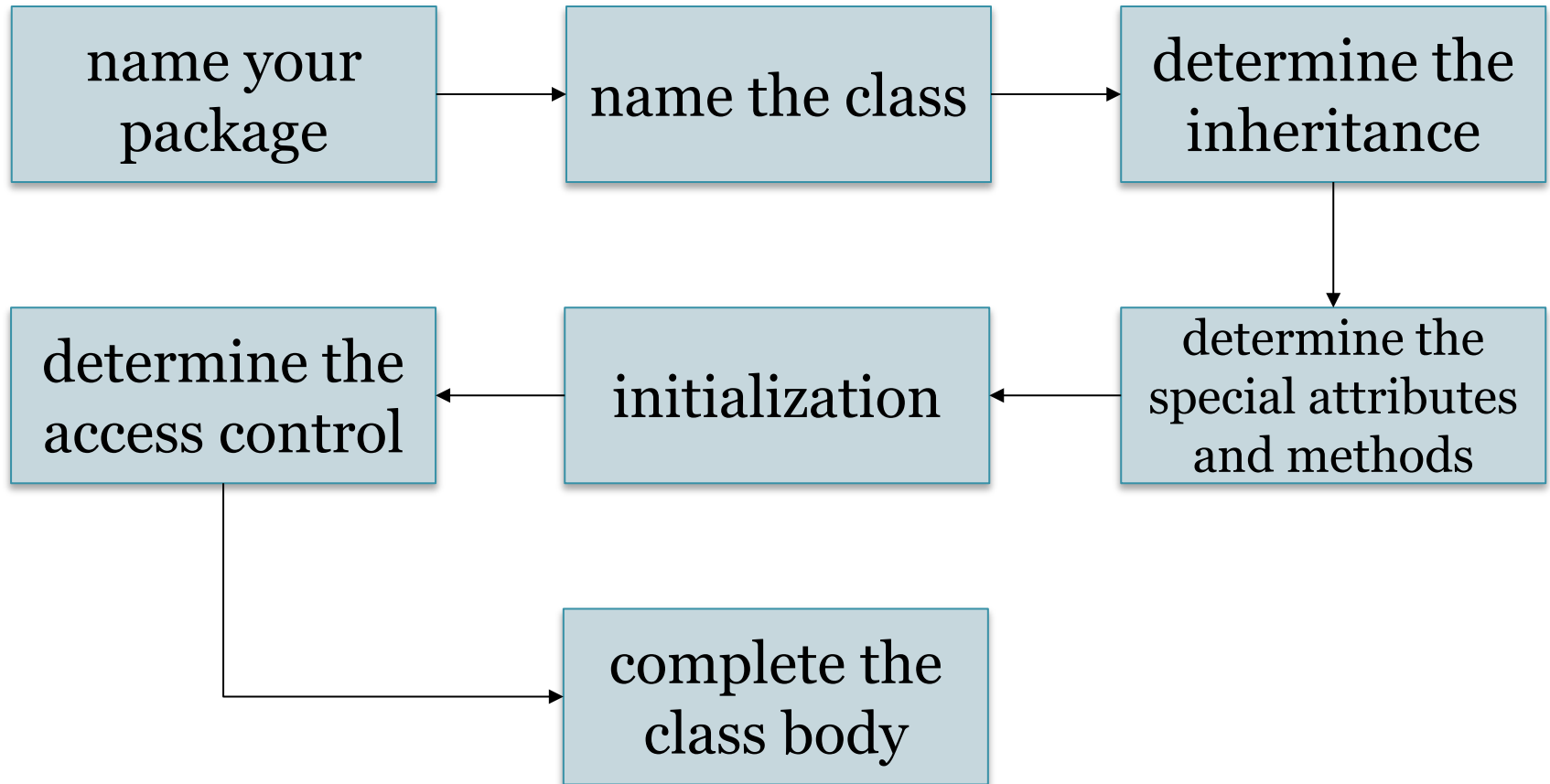
Open Discussion: Why?

Class and Object

- Class
  - Often describes a hierarchical concepts, such as: Person、Bird、Order
  - A class usually has some attributes and behaviors:
    - Attributes are called Fields, such as the age of a Person
    - Most attributes character the difference between objects, but some attributes are common, such as each Person has two legs, this kind of shared attributes is called Static attributes
    - Behaviors are called Methods, and methods can be static too.
  - IN Java, the hierarchy of classes is a tree
- Object
  - Instances of class. Such as a Person called tom, a octopus called paul...

```
name your package  →  name the class  →  determine the inheritance
                                                     ↓
determine the access control  ←  initialization  ←  determine the special attributes and methods
        ↓
complete the class body
```

A Simple Class

```java
package cn.edu.seu.cose.javacourse.ch02;
public class Person{
    private String name;
    private int age;
    public Person(String name, int age){
        this.name = name;
        this.age = age;
    }
    public void greet(){
        System.out.println("Hello, I am" + name)
            + " , and I am " + age + " years old.";
    }
    public static void main(String[] args){
        Person tom = new Person("Tom", 18);
        tom.greet();
    }
}
```

Class Components

- Package name/ Class name

- import

- Members
  - Field - static / non-static
  - Method - static / non-static

- Access Modifier(Class / Field / Method)
  - public / abstract / final
  - public / protected / private

Class and Object – Package

- Package is a set of Classes
  - To avoid classes with same names
  - To manage classes
- Define a package
  - package javacourse;
  - package cn.edu.seu.cose.javacourse;
- Import a package or a class
  - import java.io.*;
  - import java.io.File;

Class and Object – Field

- Define a Field

  ○ Access Modifier

  ○ Static Modifier (Optional)

  ○ Type

  ○ Name

  ```
  public int age;
  ```

- Non-static

  ```
  public class Person{
      public int age;
  }
  …
  Person tom = new Person();
  tom.age = 18;
  System.out.println(tom.age);
  ```

- Static   **Any bugs?**

  ```
  public class Person{
      public static int counter =0;
  }
  …
  Person tom = new Person();
  Person.counter++;
  System.out.println(Person.counter);
  ```

```java
private String name;
private int age;
public static int counter = 0;

public Person(String name, int age){
    this.name = name;
    this.age = age;
    counter++;
}

public void greet(){
    System.out.println("Hello, I am " + name
            + " , and I am " + age + " years old");
}


public static void main(String[] args){
    for(int i=0; i<10; i++){
        Person tom = new Person("Tom", 0);
        tom.greet();
    }
    System.out.println(Person.counter);
}
```

Any bugs?

the correct code

```java
private String name;
private int age;
public static int counter = 0;

public PersonWithFinalize(String name, int age) {
    this.name = name;
    this.age = age;
    counter++;
}

protected void finalize() {
    counter--;
}

public void greet() {
    System.out.println("...");
}

public static void main(String[] args) {
    for(int i=0; i<10; i++) {
        PersonWithFinalize tom = new
                PersonWithFinalize("Tom", 18);
        tom.greet();
    }
    System.out.println(PersonWithFinalize.counter);
}
```

Class and Object – Method

- ## Define a Method
  - o Access Modifier
  - o Static Modifier (Optional
  - o Return Type
  - o Name
  - o Parameter List
    (Type + Name)
  - o Method Body

```java
public class PersonWithHeight {

    public int height;

    public boolean isHigh() {
        if (this.height > 180)
            return true;
        else
            return false;
    }

    public boolean higherThan(PersonWithHeight
            someone) {
        if (this.height > someone.height)
            return true;
        else
            return false;
    }
}
```

Can this method be static ??

Can this method be static ??

Anything wrong with this class?

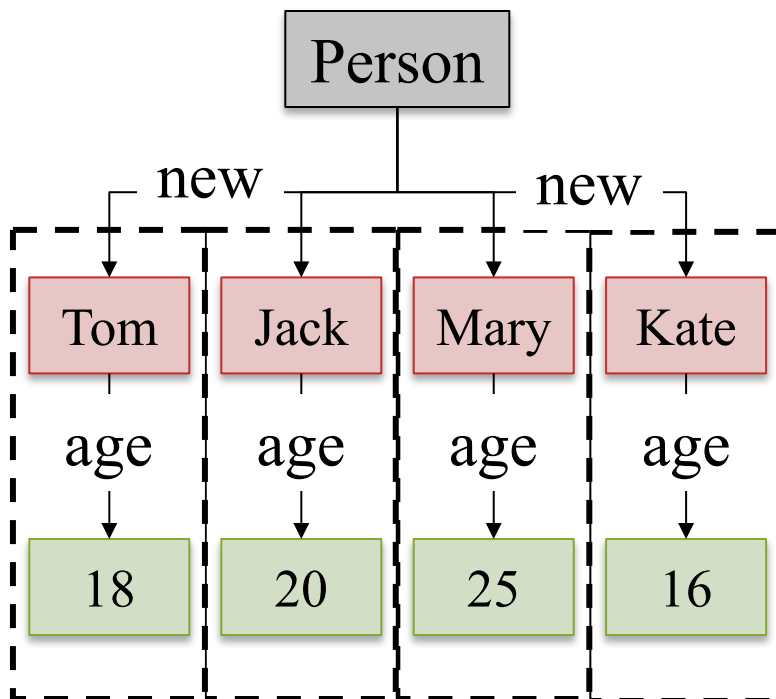# Class and Object – Method

- Static Method

```java
public class Calculator{
    public static int add(int a, int b){
        return a+b;
    }
}
...
System.out.println(Calculator.add(1 + 2));
```
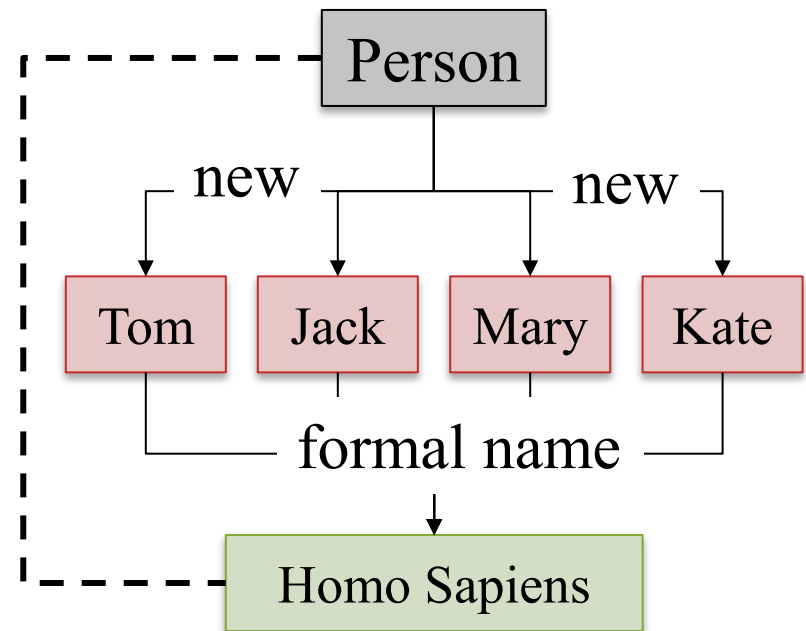
More About Static – Static Variable

Non-static Variable
(Object-level Variable)

Static Variable
(Class-level Variable)

Variables are not shared, and
memory-stored with objects.
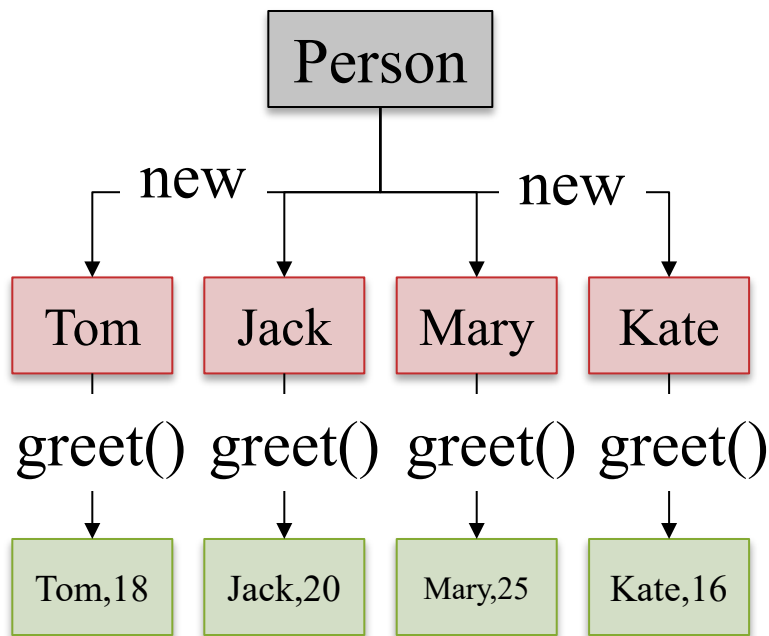
Variables are shared, and
memory-stored with class.

More About Static – Static Method

Non-static Method (Object-level Method)	Static Variable (Class-level Method)

Person

new   new

| Tom | Jack | Mary | Kate |

greet() greet() greet() greet()

| Tom,18 | Jack,20 | Mary,25 | Kate,16 |

Non-static methods are allowed to visit non-static variables. Each person may have different behavior.

Person

new   new

| Tom | Jack | Mary | Kate |

sneeze()
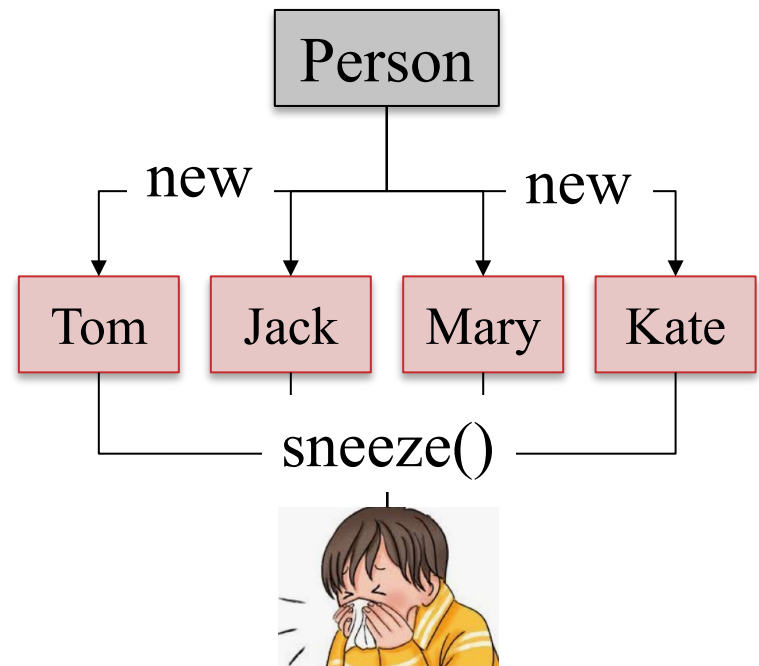
Static methods are not allowed to visit non-static variables. Each person has the same behavior.

Lab Work 1

- Create a StudentInfoSystem (SIS in short) class

  - With an INNER class Student, which defines the id, name, gender and javaScore(int);

  - In SIS，create a class with several students (max=20);

  - Using a static counter and getCounter() to count #students

  - Using a scanner and a menu to input student information;

  - Using a Student[] to store all the information;

  - Rank the list of students according to their javaScore;

  - Print out the ranked list of students

2009

```java
public class StudentInfoSystem {

    class Student{
        private int id;
        private String name;
        private boolean gender;
        private int javaScore;

        // getters and setters..
    }

    private static int counter = 0;
    private int getCounter() {
        return counter;
    }

    // more methods goes here..

    public static void main() {
        // something goes here..
    }
}
```

Class and Object – Overloading

- Method Overloading(重载)
  - Method Name
  - Method Signature
    - Method name
    - Number of Parameters
    - Types of Parameters
  - Multiple methods with same name in a class: OK (Overloading)
  - Multiple methods with same signature in a class: No!
  - Signature does not include return type, because signature reflects the specification of behavior, not the result of behavior.

Class and Object – Overloading

- Examples:

```java
public String test(String a, int b) {...}
// a method.
public void test(String s, int i) {...}
// Error! Duplicated Methods.
public String test(int a, String b) {...}
// Overloading.
public String test(String a, int b, int c) {...}
// Overloading.
```
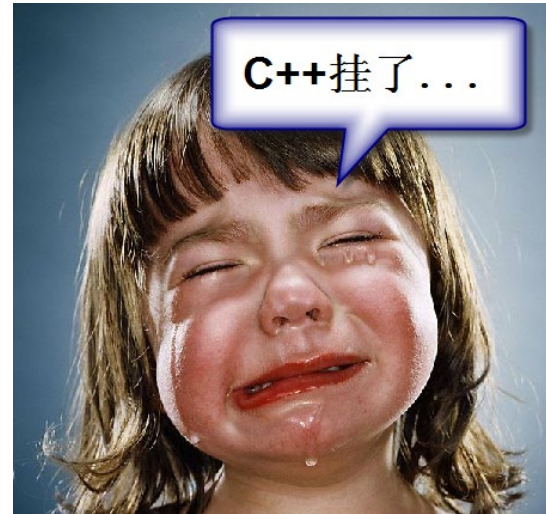
Class and Object – Parameters

- Forget them:
  - Formal Parameter?
  - Actual Parameter?
  - Pass by Value?
  - Pass by Reference?

- In Java, the Copy of Parameter is passed.

- What is copied?
  - For primary types, their value is copied
  - For objects, the reference is copied. (What is a reference?)

- Try:

```
public class ParameterPassing {
    public static void changeInt(int innerInt){
        innerInt += 10;
    }
    public static void main(String[] args){
        int i = 5;
        ParameterPassing.changeInt(i);
        System.out.println(i);
    }
}
```

- Try again:

```java
public class Person {
    public String name;
    public Person(String name){
        this.name = name;
    }
    public static void changeName(Person jerry){
        jerry.name = "Jerry";
        jerry = null;
    }
    public static void main(String[] args){
        Person tom = new Person("Tom");
        Person.changeName(tom);
        System.out.println(tom==null);
        if(tom!=null){
            System.out.println(tom.name);
        }
    }
}
```
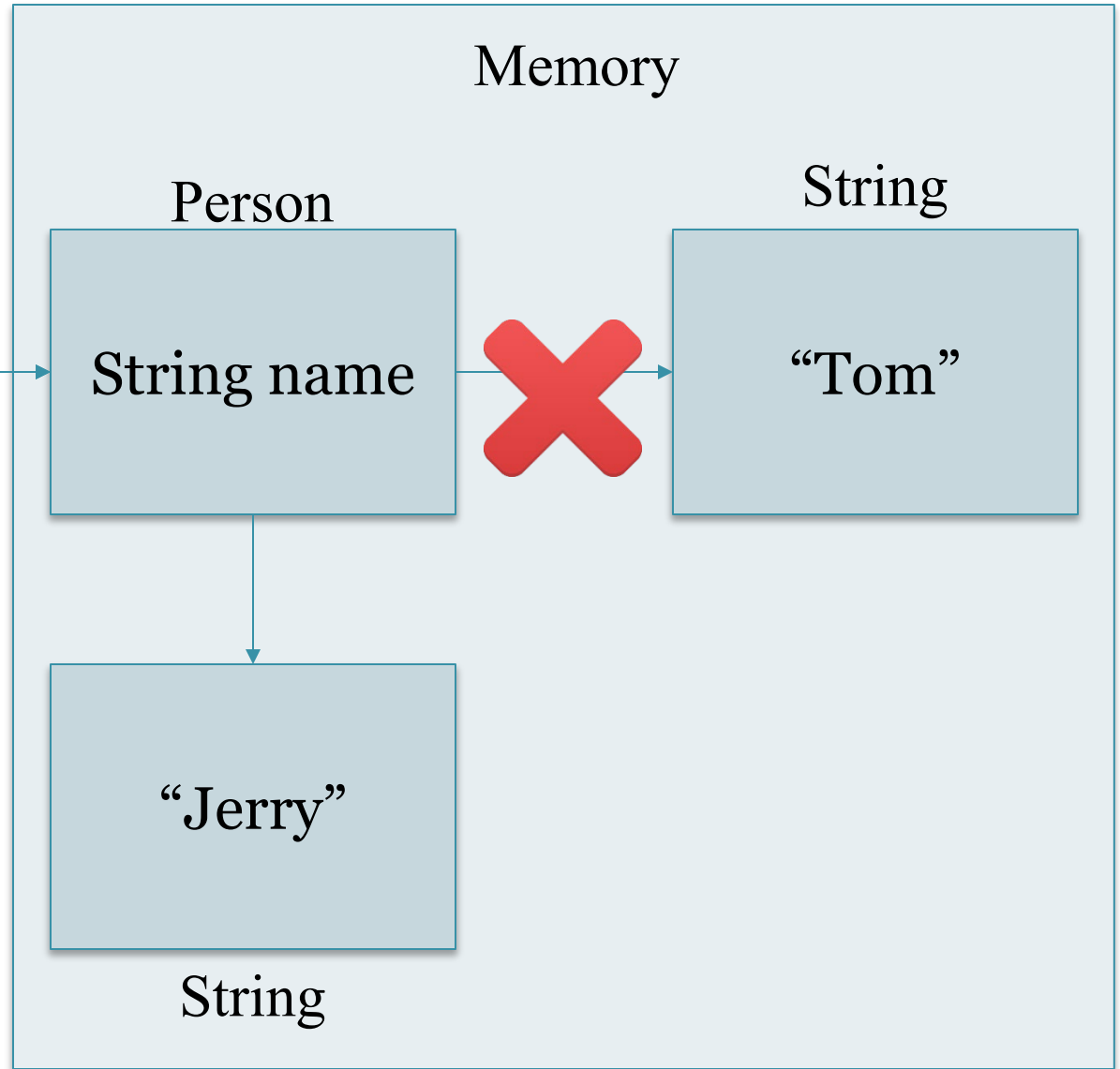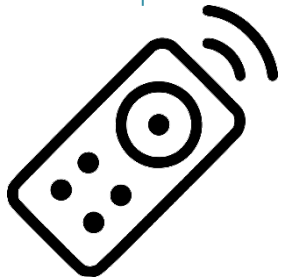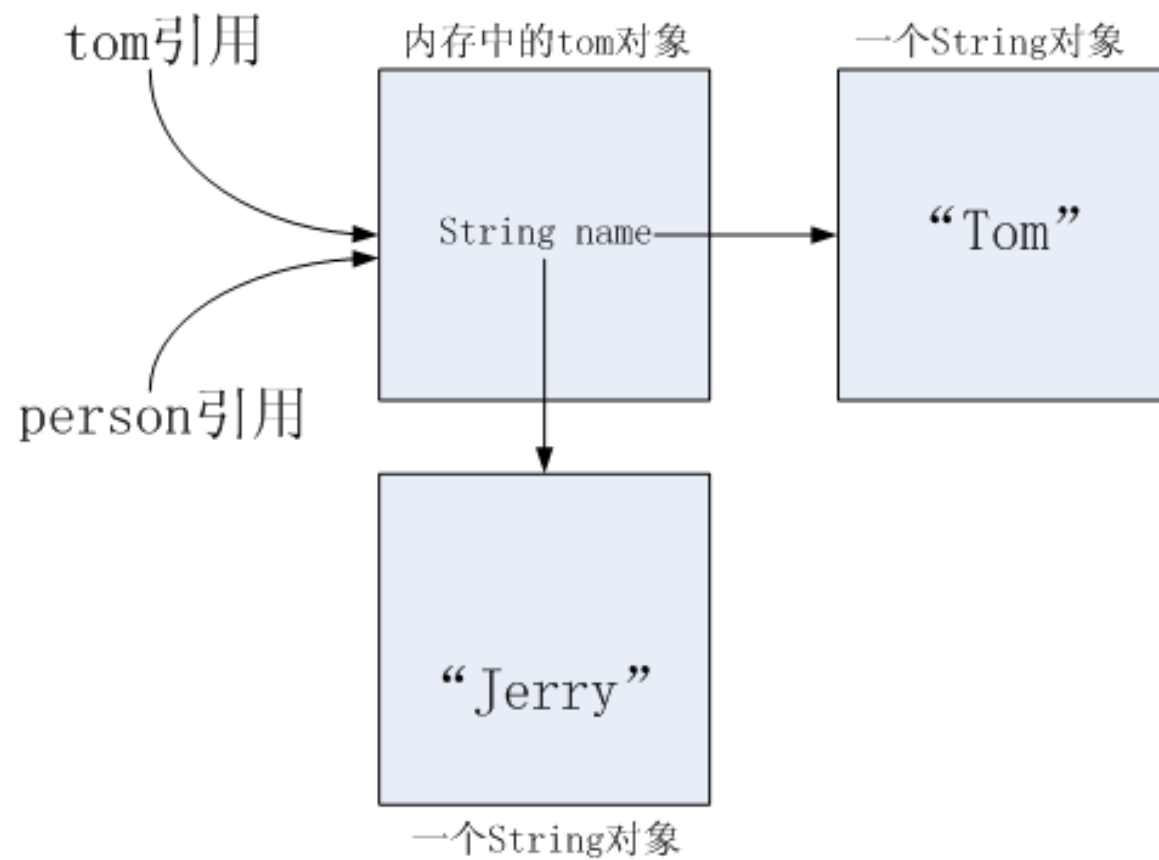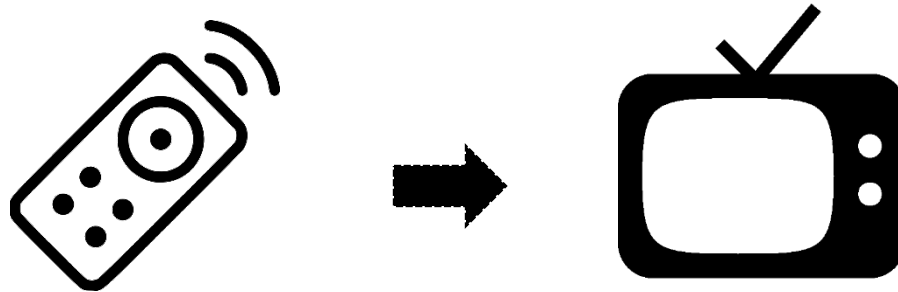
`Person tom = new Person("Tom");`

Difference between Reference and Pointer

- Pointer
  - can point to any address in the memory
  - can be used to modify the data in that address

```
student_t* p_addr = &stu1;
student_t* p_addr = 0x12000;      ←————— 引用做不到
student_t* p_addr = NULL;
```

- Reference
  - can only point to an object (or null)
  - if not null, can only modify the member attributes of that object, or invoke "." operation

- Difference
  - reference is less effective, less flexible, but much safer

- Methods with variable number of parameters

```java
public class Calculator{
    public static int add(int ...numbers){
        int result = 0;
        for(int i=0; i<numbers.length; i++){
            result += numbers[i];
        }
        return result;
    }
    public static void main(String[] args){
        System.out.println(Calculator.add(10,11));
        System.out.println(Calculator.add(10,11,12));
    }
}
```

Self-study

```
public static void add(int... numbers) {}

public void add(int a, int b, int c) {}

public void add(int[] a) {}
```

Can these three statements co-exist in the same class?

Class and Object – main method

- Each class can have zero or one main

- The main method indicates the entrance of execution

- Each main method looks like this:

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

Class and Object – Object

- Declare a reference of an object, but not create it

```
String s;   Person tom;
```

- Declare a reference of an object, and create the object

```
String s = "Hello, World";

String s = new String("Hello, World");

Person tom = new Person("Tom", 18);
```

- Null reference: Person tom = null

- Security: Reference >> Pointer

- Storage of Objects
  - Registers - inside the processors
  - Stack - object reference, primary types (栈)
  - Heap – object themselves (堆)
  - Method Area – methods, static data
  - Constant Pool
  - Non-RAM
    - Streamed Object
    - Persistent Object

- Destroying Object

  o Java GC (Garbage Collection)

  o finalize();

  o Try:

```java
public static void main(String[] args){
    System.gc();
    System.out.println("Memoery: " + Runtime.getRuntime().freeMemory());
    System.out.println("Creating houses...");
    ArrayList<House> area = new ArrayList<House>();
    for(int i=0; i<10; i++){
        area.add(new House());
    }
    System.out.println("Memoery: " + Runtime.getRuntime().freeMemory());
    System.out.println("Colleting garbage...");
    System.gc();
    System.out.println("Memoery: " + Runtime.getRuntime().freeMemory());
}
```

VM arguments:

-xms100m -Xmx100m

Class and Object – Object

- All classes in Java inherits java.lang.Object

- All objects in Java have following methods:

```java
public boolean equals(Object obj)
public int hashCode()
protected Object clone() throws CloneNotSupportedException
public final Class<?> getClass()
protected void finalize() throws Throwable
public String toString()
```

toString()

```java
public static void main(String[] args){
        Person tom = new Person("Tom", 0);
        System.out.println(tom);
        tom.greet();
}
```

1) What will happen?
2) How to print the name of Tom?

```java
public static void main(String[] args){
        Person tom = new Person("Tom", 0);
        System.out.println(tom);
        tom.greet();
}
```

// output the name of tom

Lab Work 3

- Student
  - id //学号
  - name //姓名

- Requirement
  - Create two students a and b with same id;
  - Use a.equals(b) to identify if a is the same student like b;
  - You need to overwrite .equals() method;

Comparing Two Strings

```java
String name1 = "Tom";
String name2 = "Tom";
String name3 = new String("Tom");
System.out.println(name1 == name2);
System.out.println(name1 == name3);
System.out.println(name1.equals(name3));
```

Lab Work 4

○ Try to print out .hashCode() of a certain object;

Reference

- Inside The Java Virtual Machine (深入浅出Java虚拟机)

Construction and Initialization

- How to describe the construction of an object in a class?

  - Constructor

    - Default Constructor

    - Constructor with parameters

  - Initialization Block (self-study)

Construction and Initialization

- **Constructor**
  - Default
  - With Parameters

```java
public class Person(){
    public String name;
    public int age;
    public boolean isEducated;
    public Person(){
        this.isEducated = true;
    }
    public Person(String name, int age){
        this();
        this.name = name; this.age = age;
    }
    public Person(String name, int age, boolean isEducated){
        this(name, age);
        this.isEducated = isEducated;
    }
}
```

About Construction

```java
public class Person {
    private String name;
    private int age;

    public void greet(){
        System.out.println("Hello, I am " +
                name + " and I am " + age +" years old.");
    }

    public static void main(String[] args) {
        Person tom = new Person();
        tom.greet();
    }
}
```
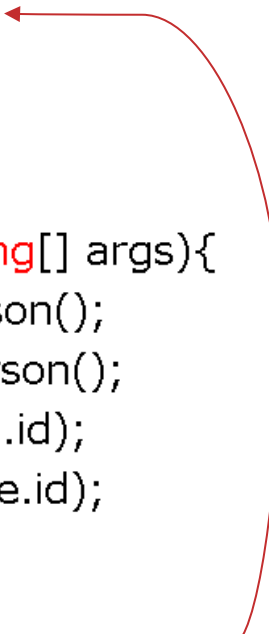
Is this compilable? Is this runnable? If yes, What is the result?

- Initialization

Block

```
public class Person{
    public int id;
    public static int counter;
    {
        id = counter++;
    }
    public static void main(String[] args){
        Person tom = new Person();
        Person mike = new Person();
        System.out.println(tom.id);
        System.out.println(mike.id);
    }
}
```

Attention: something is not correct in this code

Self-study

- Static

  Initialization

  Block

```java
public class Person{
    public int id;
    public static int counter;
    public static int getBeginID(){
        ...   //  Get initial ID from database
    }
    static{
        counter = getBeginID();
    }
    ...
}
```

Self-study

- Think:

  ○ Why use initialization blocks?

  ○ What is the difference between initialization blocks and static initialization blocks?

Class and Object – Access Control

- # Why Do We Need Access Control?
  - Encapsulation
  - Data Hiding

- # Without Access Control:
  - Debugging becomes difficult
  - Data and programs become unsafe

Class and Object – Access Control

- AC Modifier for Classes
  - *default*
  - public

- AC Modifier for Members
  - *default (package)*
  - public
  - private
  - protected

Class and Object – Access Control

	Same class	Same package	Subclass in different package	Non-Subclass in different package
public	OK	OK	OK	OK
protected	OK	OK	OK	NO
default(package)	OK	OK	NO	NO
private	OK	NO	NO	NO

- Getter and Setter Methods

```
…
private String name;
private int age;
public String getName() {return name;}
public void setName(String name) {this.name = name;}
public int getAge() {return age;}
public void setAge(int age) {
    if(age>150 || age<0){
        age = 0;
        System.out.println("Wrong age!");
    }else{
        this.age = age;
    }
}
…
```

Self-study

- Data, Information and Knowledge

- Non-structural, semi-structural and structural data

- XML
  - XML and XML Schema
  - XML vs. HTML
  - * Ant

```xml
<?xml version="1.0" encoding="utf-8"?>
<Person id="x.zhang">
    <firstname>Xiang</firstname>
    <lastname>Zhang</lastname>
    <organization>
        <College id="COSE">
            College of Software Engineering
        </College>
    </organization>
<Person>
```

Lab Work 5 – Student Management System

- Two classes: System and Student

  - System: input (create students/ modify students) and output (show statistics)

  - Student: getter and setters of each student (name, age, score)

- Modify Student Information using name;

- Maximum 50 students

```
Student[] students = new Student[50];
```

- Statistics: Average Score/ Max / Min

```
Problems  @ Javadoc  Declaration  Console ✕

StudentManagementSystem [Java Application] C:\Pro
Welcome to the Student Management System.
Please select:
1. Create a New Student.
2. Modify an Existed Student.
3. Show Statistics.
1
Please input the information of a student.
Please input the name of the student:Zhang
Please input the age of the student:18
Please input the score of the student:99
Succesfully created a new student.
Please select:
1. Create a New Student.
2. Modify an Existed Student.
3. Show Statistics.
1
Please input the information of a student.
Please input the name of the student:Wang
Please input the age of the student:18
Please input the score of the student:90
Succesfully created a new student.
Please select:
1. Create a New Student.
2. Modify an Existed Student.
3. Show Statistics.
3
The average score: 94
Please select:
1. Create a New Student.
2. Modify an Existed Student.
3. Show Statistics.
```

Forecast

- Abstraction
  - Abstract Class
  - Interface
- Inheritance
- Polymorphism