Advanced Java Programming
Chapter 2

Object Oriented Programming

Zan Wang（王赞）
Office: 55-A413
Email: wangzan@tju.edu.cn

2020

Outline

- Basic concepts of OOP
  - Class and Object
  - Encapsulation, Inherent and Polymorphism
  - Some related keywords
    - Main
    - New
    - This
    - Static
    - Final
  - Interface and Abstract Class

BASIC CONCEPT OF OOP

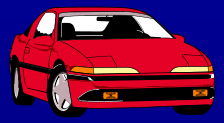
The Object-Oriented (OO) Programming Paradigm

- Object-Oriented Programming (OOP) is one of the programming paradigms in computer science

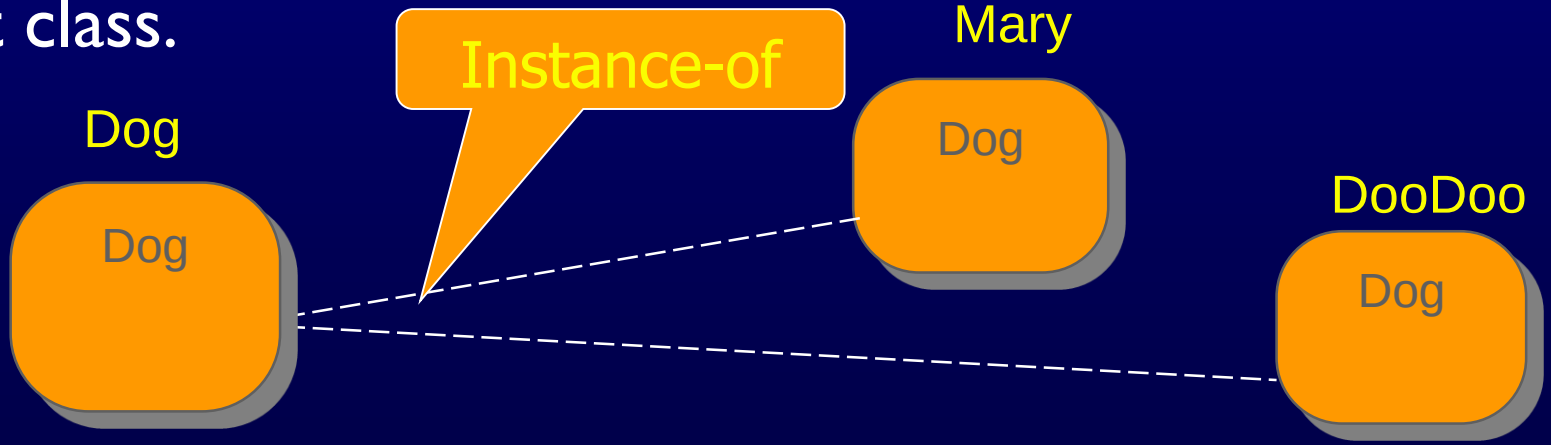
- OOP is the dominant modern programming paradigm
  - Better concepts and tools to model and represent the real world as closely as possible
    - Model of reality
    - Behavior modeling
  - Better reusability & extensibility (inheritance)
    - Reduce the time/cost of development
  - Enhanced maintainability & improved reliability – "Encapsulation" and "Information Hiding"
    - Object only accessible through the external interface
    - Internal implementation details are not visible outside
    - Localized changes to implementation of classes
    - Completeness: all required operations are defined
    - Independent testing and maintenance

Classes and Objects

- An *object* is a thing

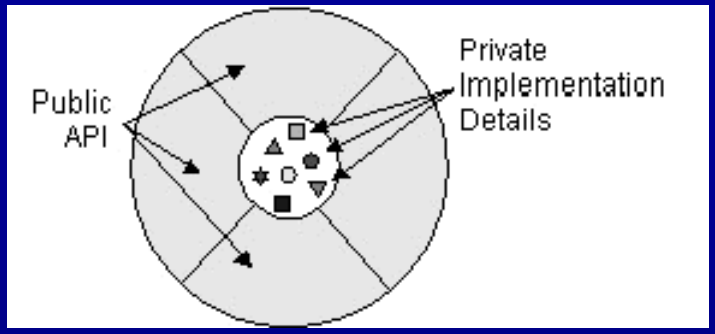




- A *class* (e.g., Dog) is a kind of mold or template to create objects (e.g., Mary)
- An object is an *instance* of a class. The object *belongs* to that class.



Classes and Objects

- An object has ***behaviors***
  - Data
  - Methods that manipulate that data



- An object has ***states***
  - The data represent the state of the object
  - Data can also describe the relationships between this object and other objects
  - Example: A CheckingAccount might have
    - A balance (the internal state of the account)
    - An owner (some object representing a person)
- Definition: An object is a software bundle of ***variables*** and related ***methods*** (function).

Objects vs. Classes

- Objects:
  - E.g., a real car.
- Classes: or types
  - E.g., Sedan.

A Class is a blueprint or template of some objects

Object is an Instance of some class.

Classes in Java

- A *class* -- is a template that describes the data and behavior associated with instances of that class.
- An *object* is an instance of some class.
- The *data* associated with a *class* or *object* is stored in variables.
- The *behavior* associated with a class or object is implemented with methods.
  - Methods are similar to the functions or procedures in C.

Definition of Class

```
[Access Modifier] class <class name> {
    [Access Modifier] <Data Type> <Variable Name> ;
    …

    [Access Modifier] <Return Data Type> <Method Name> (Parameters….)
     { /* ... */ }
    …
}
```

Class Car in Java

- Example 2.1

```
package cn.tju.scs.c01;
public class Car {
        String name;
        double price;
        void getCarInfo(){
                System.out.println("汽车名称： "+ name + "，汽车价格： " + price);
        }
}
```

```
package cn.tju.scs.c01;
public class TestMain {
    public static void main(String[] args) {
        Car car = new Car();
        car.name = "大众";
        car.price = 20.5;
        car.getCarInfo();
    }
}
```

Some foundations of Java

- main()
  - A Java application must contain a **main()** method whose signature looks like this

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

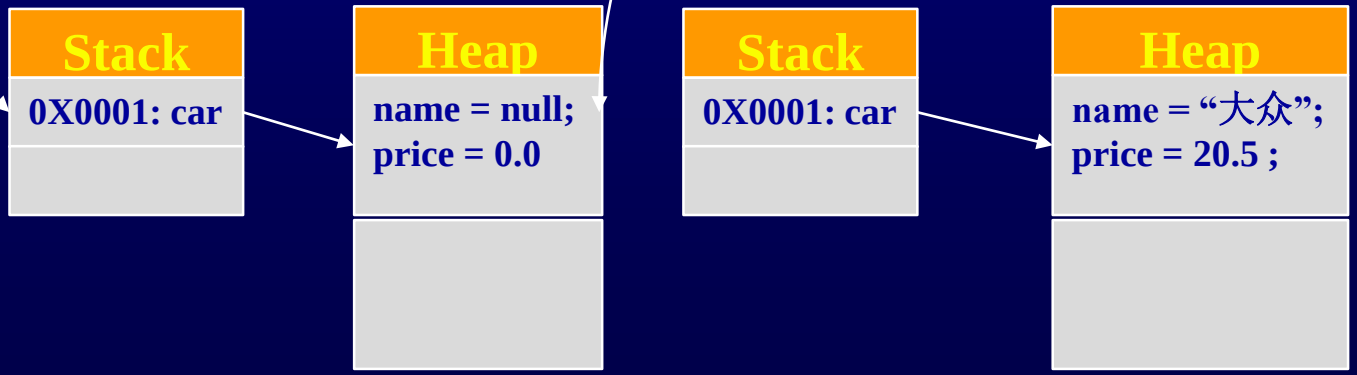
    - public indicates that the main() method can be called by any object
    - static indicates that the main() method is a class method.
    - void indicates that the main() method has no return value.
    - The main() method accepts a single argument: String[] args. This array of Strings is the mechanism through which the runtime system passes information to your application. Each String in the array is called a command line argument. Command line arguments let users affect the operation of the application without recompiling it.

Some foundations of Java

- new
  - A class provides the blueprint for objects; you create an object from a class.

```
Car car = new Car();
```

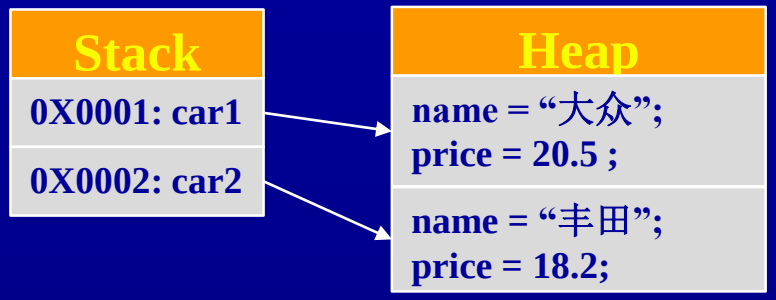
- **Declaration**: The code set in orange color are variable declaration that associate a variable name with an object type.
- **Instantiation**: The new keyword is a Java operator that creates the object.
- **Initialization**: The new operator is followed by a call to a constructor, which initializes the new object.

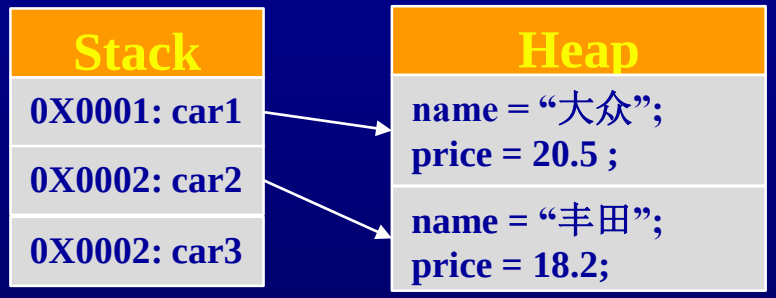


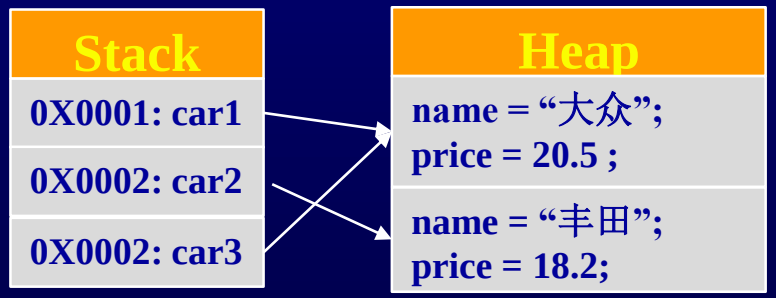
Reference Analysis

- Example 2.2

```
public class TestMainRef {
    public static void main(String[] args) {
        Car car1 = new Car();
        Car car2 = new Car();
        car1.name = "大众";
        car1.price = 20.5;
        car2.name = "丰田";
        car2.price = 18.2;
        car1.getCarInfo();
        car2.getCarInfo();
        Car car3 = null;
        car3 = car1;
        car3.price = 22.2;
        System.out.println("----------------");
        car1.getCarInfo();
        car3.getCarInfo();
    }
}
```







Constructor

- Example 2.2

```
Car car = new Car();
```

  - Constructor
    - A class contains constructors that are invoked to create objects from the class blueprint.
    - Constructor declarations look like method declarations—except that they use the name of the class and have no return type.
    - The compiler automatically provides a no-argument, default constructor for any class without constructors.

Constructor Overload

- Method Overloading
  - If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.
  - Method overloading increases the readability of the program.
  - There are two ways to overload the method in java
    - By changing number of arguments
    - By changing the data type
  - Some questions:
    - Is it possible to overload by changing the return type of the method only?
    - Can we overload java main() method?

Back to Example2.1

- Example 2.1 – 2.2

```
package cn.tju.scs.c01;
public class Car {
        String name;
        double price;
        void getCarInfo(){
                System.out.println("汽车名称："+ name + "，汽车价格：" + price);
        }
}
```

```
package cn.tju.scs.c01;
public class TestMain {
    public static void main(String[] args) {
        Car car = new Car();
        car.name = "大众";
        car.price = 20.5;
        car.getCarInfo();
    }
}
```

Encapsulation (封装)

- Hide the object's nucleus from other objects in the program.
  - The implementation details can change at any time without affecting other parts of the program.
- It is an ideal representation of an object, but
  - For implementation or efficiency reasons, an object may wish to expose some of its variables or hide some of its methods.
- Benefits:
  - Modularity
    - The source code for an object can be written and maintained independently of the source code for other objects.
  - Information hiding
    - An object has a public interface that other objects can use to communicate with it.

Getter and Setter

- getName(), setName() / getPrice(), setPrice()
- Example 2.3

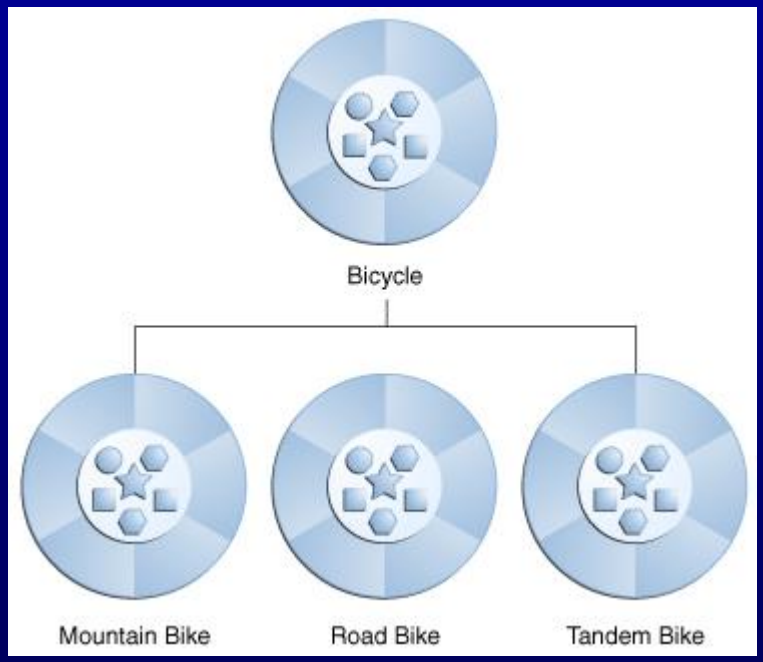
```
public class CarEn {
        private String name;
        private double price;
        void getCarInfo(){
                System.out.println("汽车名称："+ getName() + "，汽车价格：" + getPrice());
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public double getPrice() {
                return price;
        }
        public void setPrice(double price) {
                if (price > 0){
                        this.price = price;
                }
        }
}
```

this

- Within an instance method or a constructor, this is a reference to the ***current object***
  - Using *this* with a Field
  - Using *this* with a Constructor
    - First line!!
  - Question
    - Why to use this?
  - See example 2.3 ***CarThis.java / CarThisMain.java***

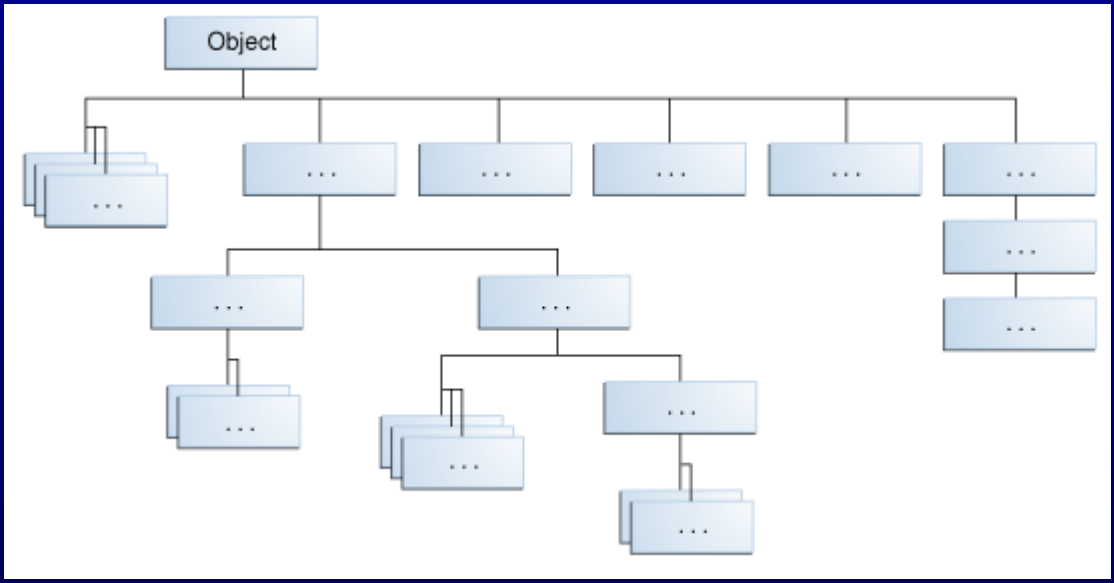
Inheritance

- Subclasses inherit variables and methods from superclass's.
  - States:
    - Gear & Speed
  - Behaviors:
    - Brake & Accelerate
- Subclasses can
  - Add variables and methods.
  - Override inherited methods.
- Benefits:
  - Provide specialized behaviors
  - Reuse the code in the superclass
  - Abstract classes -- define "generic" behaviors.



Inheritance

- Object , which has no superclass, every class has one and only one direct superclass (single inheritance). In the absence of any other explicit superclass, every class is implicitly superclass of Object.



Mankind extends Animal

```
public class Animal {
    private int height = 0;
    private int weight = 0;
    public void talk( ) { System.out.println("Arhh");}
}

public class Mankind extends Animal {
    private int iq = 120;
    public void talk( ) { System.out.println("Hello");}
}
```

Should be in different files

One Java file can only have one public class

Mankind is-a Animal

Details for Inheritance

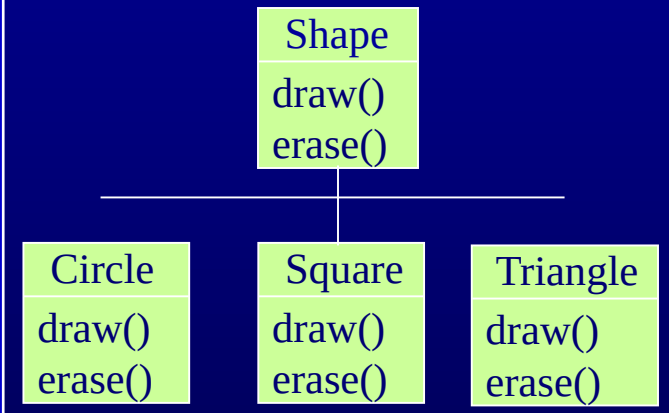
- The inherited fields can be used directly, just like any other fields.
- You can declare a field in the subclass with the same name as the one in the superclass, thus hiding it (not recommended) D.
- You can declare new fields in the subclass that are not in the superclass. E
- The inherited methods can be used directly as they are. D
- You can write a new instance method in the subclass that has the same signature as the one in the superclass, thus overriding it.
- You can write a new static method in the subclass that has the same signature as the one in the superclass, thus hiding it.
- You can declare new methods in the subclass that are not in the superclass.
- You can write a subclass constructor that invokes the constructor of the superclass, either implicitly or by using the keyword super.

Polymorphism

- Polymorphism: In Java, the principle that the actual type of the object determines the method to be called is called polymorphism
- 事物在运行过程中存在不同的状态

```
class Shape {
    void draw( ) {}
    void erase( ) {}
}

class Circle extends Shape {
   void draw( ) {
            System.out.println("Circle.draw( ) ");
   }
   void erase( ) {System.out.println("Circle.erase( )");}
}
```



Polymorphism

- ***See example 2.4 PolyMorphDemo.java***

- Upcasting（向上转型）：参数的统一
  - Class A
  - Class B extends A
  - A a = new B();
- Downcasting（向下转型）：父类调用子类特殊的方法
  - B b = (B)a;

- instanceof

```
class Animal{
	int num = 10;
	static int age = 20;
	public void eat(){
		System.out.println("动物吃饭");
	}
	public static void sleep(){
		System.out.println("动物睡觉");
	}

	public void run(){
		System.out.println("动物在跑");
	}
}
```

```
class Cat extends Animal{
int num = 80;
static int age = 90;
String name = "tomcat";

public void eat(){
  System.out.println("猫吃饭");
}
public static void sleep(){
	System.out.println("猫睡觉");
}

public void catchMouse(){
	System.out.println("猫在抓老鼠");
}
}
```

```
public class PolyMor {
    public static void main(String[] args) {
		Animal am = new Cat();
		am.eat();
		am.sleep();
		am.run();
		System.out.println(am.num);
		System.out.println(am.age);

    }

}
```

```
public class PolyMor {
    public static void main(String[] args) {
		Animal am = new Cat();
		am.eat();
		am.sleep();
		am.run();
		System.out.println(am.num);
		System.out.println(am.age);
		System.out.println("--------------------");
		Cat ct = (Cat)am;
		ct.eat();
		ct.catchMouse();
		ct.run();
		ct.sleep();
		System.out.println(ct.name);
		System.out.println(ct.num);
		System.out.println(ct.age);
    }

}
```

Static

- In Java, a *static* member is a member of a class that isn't associated with an instance of a class. Instead, **the member belongs to the class itself**. As a result, you can access the static member without first creating a class instance. The two types of static members are static fields and static methods:
  - **Static field**: A field that's declared with the static keyword. The value of a static field is the same across all instances of the class. In other words, if a class has a static field named *CompanyName*, all objects created from the class will have the same value for *CompanyName*.
  - **Static method**: A method declared with the static keyword. Like static fields, static methods are associated with the class itself, not with any particular object created from the class. As a result, you don't have to create an object from a class before you can use static methods defined by the class.

Static variable

- The problem without static variable

```
public class Student {
        int rollno;
        String name;
        String college = "SCS";
}
```

- Example 2.5 StudentStatic.java
- Example 2.6 CarStatic.java/TestMainStatic.java
- ***Exercise 2.1*** Please count the number of instances which have been created for a Object.

Static method

- Apply static keyword with any method, it is known as static method:
  - A static method belongs to the class rather than object of a class.
  - A static method can be invoked without the need for creating an instance of a class.
  - static method can access static data member and can change the value of it.
- Example 2.7 StudentStaticMethod.java
- Restrictions for static method
  - The static method can not use non static data member or call non-static method directly.
  - this and super cannot be used in static context.

Static Block

- Is used to initialize the static data member.
- It is executed before main method at the time of class loading.
- Example 2.8 StaticBlockDemo.java
- Question: Can we execute a program without main() method?

Final

- The Final keyword in java can be used in many context. Final can be :
  - Variable
    - Stop value change
  - Method
    - Stop method overriding
  - Class
    - Stop inheritance

Final Variable, method and Class

- Final Variable
  - If you make any variable as final, you cannot change the value of final variable(It will be constant)
  - Example of final variable. **Example 2.9 BikeFinalVar.java**
- Final Method
  - If you make any method as final, you cannot override it.
  - **Example 2.10 BikeFinalMethod.java**
- Final Class
  - If you make any class as final, you cannot extend it

Some questions about Final

- Q1: Is final method inherited?
  - Yes, final method is inherited but you cannot override it
  - See BikeFinalMethod.java
- Q2: What is blank or uninitialized final variable?
  - A final variable that is not initialized at the time of declaration is known as blank final variable.
  - If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed, it is useful.
  - It can be initialized only in constructor.
- static blank final variable
  - A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

Abstraction in Java

- Abstraction is a process of hiding the implementation details and showing only functionality to the user.
- Abstraction lets you focus on what the object does instead of how it does.
- Ways to achieve Abstraction
  - Abstract class
  - Interface

Abstract class

- A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body).

Abstract class

- A class that is declared as abstract is known as abstract class. It needs to be extended and its method implemented. It cannot be instantiated.

```
abstract class A{}
```

- Abstract method:
  - A method that is declared as abstract and does not have implementation is known as abstract method.

```
abstract void printstatus();
```

- Abstract class that has abstract method:
  - See Example 2.11 HondaAbstract.java

The real scenario of abstract class

- A real scenario that we need a set of interfaces and a couple of default methods.
- Example 2.12: Factory method ACFactory.java
- An abstract class cannot be instantiated, but you can create a concrete class based on an abstract class, which then can be instantiated. To do so you have to inherit from the abstract class and override the abstract methods, i.e. implement them.

Template Design

- Three kinds of object
  - Robot: charge()[eat()], work()
  - Person: eat(), work(), sleep()
  - Dog: eat(), sleep()
- Implements the action of the robot, person and dog

Interface

- An interface in java is a blueprint of a class. It has ***static constants*** and ***abstract methods***.
- It is used to achieve abstraction and multiple inheritance in Java.
- Interfaces can have methods and variables but the methods declared in interface contain only method signature, not body

Why and how to use Java interface?

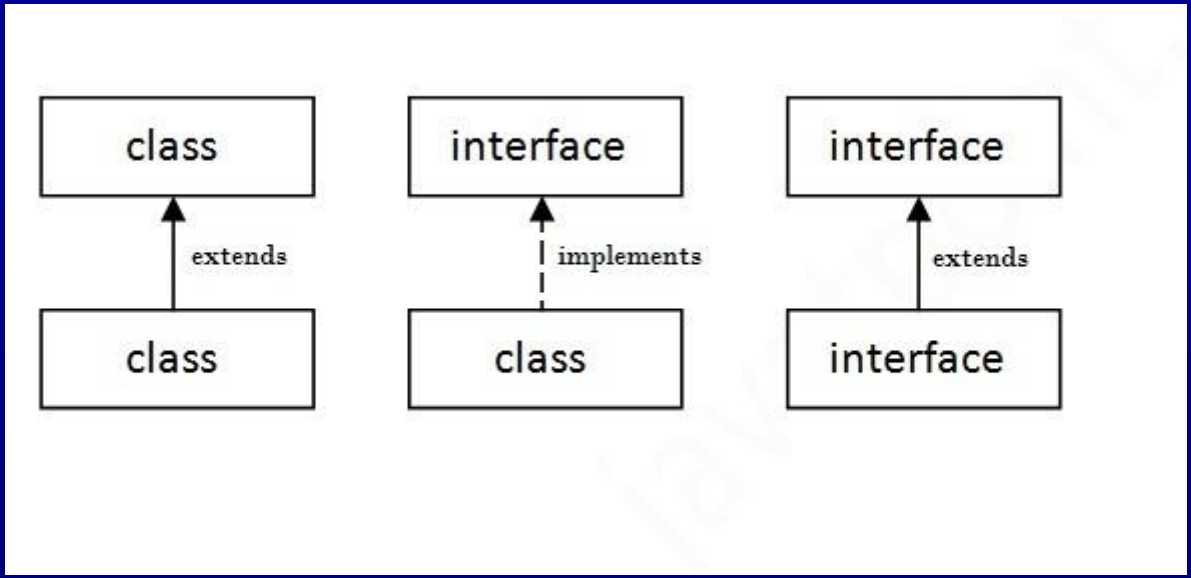
- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

```
interface <interface_name>{

    // declare constant fields
    // declare methods that abstract
    // by default.
}
```

- Interface fields are public, static and final by default, and methods are public and abstract.

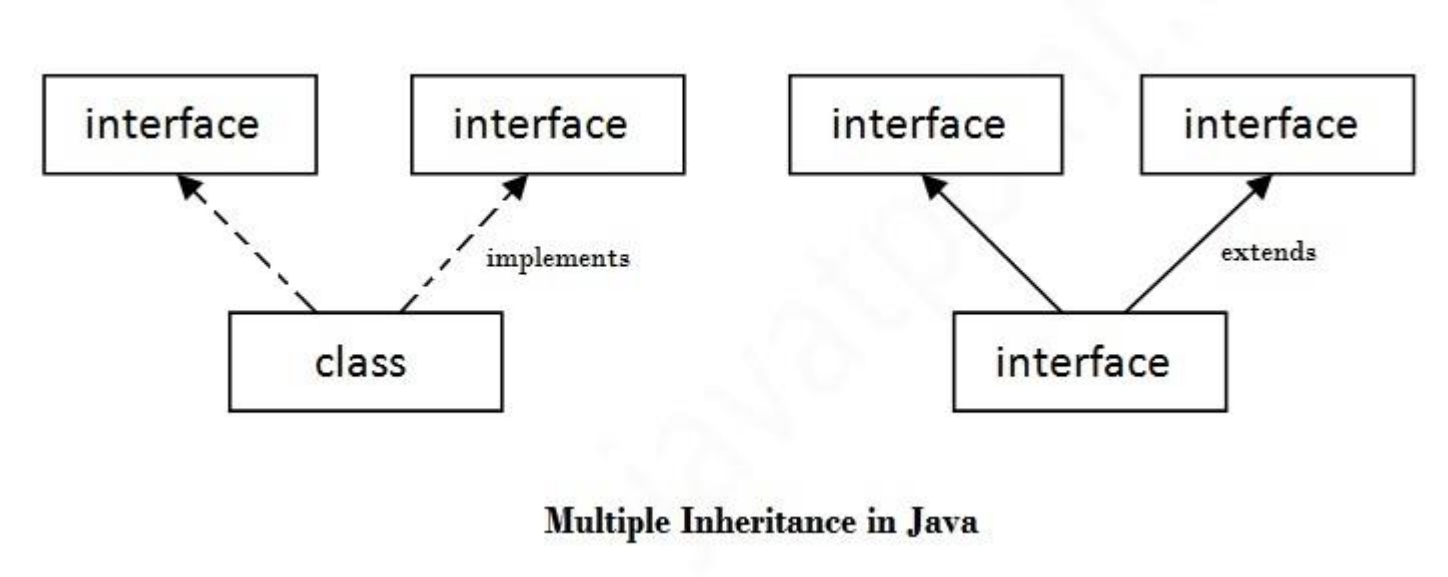
Relationship between classes and Interfaces



- *Example 2.13 InterfaceDemo.java*
- *Example 2.14 InterfaceBank.java*

Multiple inheritance by interface

- If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.



- See Example 2.15 InterfaceMulti.java

Interface (cont.)

- Interface inheritance
  - A class implements interface but one interface may extend another interface .
  - ***See Example 2.16 InterfaceExtends.java***
- Interface in Java 8
  - Since Java 8, we can have method body in interface. But we need to make it default method.
  - ***See Example 2.17 InterfaceDefault.java***
  - ***See Example 2.18 InterfaceConflict.java***
- What is marker or tagged interface?
  - An interface which has no member is known as marker or tagged interface. For example: *Serializable*, *Cloneable*, Remote etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

Factory Design Pattern

- First: See ***Example 2.19 FactoryDemo.java***
- ***FactoryDraw.java***
- ***FactoryReflect.java***

Abstract Class vs. Interface

- Similar:
  - You cannot instantiate them
  - They may contain a mix of methods declared with or without an implementation.
- Not Similar
  - Fields type
    - With abstract classes, you can declare fields that are not static and final, and define public, protected, and private concrete methods.
    - With interfaces, all fields are automatically public, static, and final, and all methods that you declare or define (as default methods) are public.
  - Extends
    - You can extend only one class, whether or not it is abstract,
    - You can implement any number of interfaces.

Abstract Class vs. Interface

- Consider using abstract classes if any of these statements apply to your situation:
  - You want to share code among several closely related classes.
  - You expect that classes that extend your abstract class have many common methods or fields, or require access modifiers other than public (such as protected and private).
  - You want to declare non-static or non-final fields. This enables you to define methods that can access and modify the state of the object to which they belong.
- Consider using interfaces if any of these statements apply to your situation:
  - You expect that unrelated classes would implement your interface. For example, the interfaces Comparable and Cloneable are implemented by many unrelated classes.
  - You want to specify the behavior of a particular data type, but not concerned about who implements its behavior.
  - You want to take advantage of multiple inheritance of type.

Homework 2.1

- **Singleton pattern**. In software engineering, the singleton pattern is a software design pattern that restricts the instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system. The concept is sometimes generalized to systems that operate more efficiently when only one object exists, or that restrict the instantiation to a certain number of objects.
- The singleton design pattern describes how to solve above problems:
  - Hide the constructor of the class.
  - Define a public static operation that returns the sole instance of the class.
- Please implement the singleton pattern in Java by yourself.
- Deadline: 23:59:59, 5th March. 2020