softserve experience matters

Java Core

Inheritance

IT Academy
05/2016

Agenda

- Java OOPs Concepts
- Abstract class
- Interface
- Inheritance in Java
- Polymorphism
- this, super
- Object. Override
- Final

Java OOPs Concepts

Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

Class

Collection of objects is called class. It is a logical entity.

Inheritance

When one object acquires all the properties and behaviors of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism

When one task is performed by different ways i.e. known as polymorphism. For example: cat speaks meow, dog barks woof etc.

Java OOPs Concepts

Abstraction

Hiding internal details and showing functionality is known as an abstraction. For example: phone call, we don't know the internal processing.

- In java, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

- A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Abstract Classes

A class must be declared ***abstract*** when we need to forbid creating instances of this class.

Abstract class may has one or more ***abstract methods***.

A method is declared abstract when it has a method heading, but no body – which means that an abstract method has no implementation code inside curly braces like normal methods do.

- The derived class must provide a definition method;
- The derived class must be declared abstract itself.

A non abstract class is called a concrete class.

Abstract Classes

```
/* The Figure class must be declared as abstract because it
contains an abstract method */
public abstract class Figure {
    /* because this is an abstract
    method the body will be blank */
    public abstract double getArea();
}

public class Circle extends Figure {
    private double radius;
    public Circle (double radius) {
        this.radius = radius;}
    public double getArea() {
          return (3.14 * (radius * 2));   }
}
```

Abstract Classes

```
public class Rectangle extends Figure {
  private double length, width;

  public class Rectangle(double length, double width) {
        this.length = lengt;
        this.width = width;
  }

  public double getArea() {
         return length * width;
  }
}
```

Interfaces

- An **interface** is a reference type in Java, it is similar to class, it is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.
- Along with *abstract methods* an interface may also contain *constants*, *default methods*, *static methods*, and *nested types*. Method bodies exist only for default methods and static methods.
- An interface is essentially a **type** that can be satisfied by any class that implements the interface.
- Any class that implements an interface must satisfy 2 conditions
  - It must have the phrase "implements *Interface_Name*" at the beginning of the class definiton;
  - It must implement all of the method headings listed in the interface definition.

Interfaces

```
public interface Dog {
    public boolean barks();
    public boolean isGoldenRetriever();
}

public class SomeClass implements Dog {
    public boolean barks() {
        // method definition here
    }
    public boolean isGoldenRetriever() {
        // method definition here
    }
}
```

Default Methods for Interfaces

Java 8 enables us to add non-abstract method implementations to interfaces by utilizing the default keyword. This feature is also known as *Extension Methods*.

For example:

```
interface Formula {
    double calculate(int a);

    default double sqrt(int a) {
        return Math.sqrt(a);
    }
}
```

Default Methods for Interfaces

- Besides the abstract method calculate the interface Formula also defines the default method sqrt. Concrete classes only have to implement the abstract method calculate. The default method sqrt can be used out of the box.

```
Formula formula = new Formula() {
    @Override
    public double calculate(int a) {
   return sqrt(a * 100);
    }
};

formula.calculate(100);  // 100.0
formula.sqrt(16);        // 4.0
```

Inheritance

Assignment operator. What will be done ?

```
int num=1;
double data = 1.0;
data = num;                 //  num = data; ???

class Aclass {
    int field1 = 10;
}

class Bclass extends Aclass {
    int field2 = 20;
}

Aclass a = new Aclass( );
Bclass b = new Bclass( );
a = b;                      //  b = a; ???
```

Inheritance

```
public class Circle {
private double radius;

    // Constructors
    public Circle() {
    this.radius = 1.0;
    }
    public Circle(double radius) {
    this.radius = radius; }

    // Getters and Setters
    // Return the area of this Circle
    public double getArea() {
       return radius * radius * Math.PI;
    }
}
```

Inheritance

```
public class Cylinder extends Circle {
private double height;

    // Constructors
    public Cylinder() {
        super();  // invoke superclass' constructor Circle()
        this.height = 1.0;
    }
    public Cylinder(double height) {
        super();  // invoke superclass' constructor Circle()
        this.height = height;
    }
    public Cylinder(double height, double radius) {
        // invoke superclass' constructor Circle(radius)
       super(radius);
        this.height = height;
    }
```

Inheritance

```
    // Getter and Setter
    // Return the volume of this Cylinder
    public double getVolume() {
        // Use Circle's getArea()
        return getArea() * height;
    }

    // Describible itself
    public String toString() {
        return "This is a Cylinder";
    }
}
```

Inheritance

```
public class ClassA {
    public int i = 1;
    public void m1() {
        System.out.println("ClassA, metod m1, i = " + i);
}
    public void m2() {
        System.out.println("ClassA, metod m2, i = " + i);
}
    public void m3() {
        System.out.print("ClassA, metod m3,
                                        runnind m4():");
m4();  }
    public void m4() {
        System.out.println("ClassA, metod m4");
    }
}
```

Inheritance

```
public class ClassB extends ClassA {
     public double i = 1.1;
     public void m1() {
    System.out.println("ClassB, metod m1, i= " + i);
     }
     public void m4() {
    System.out.println("ClassB, metod m4");
     }
}
```

Automatically added default constructor.

Inheritance

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

        System.out.println("The Start.");

        ClassA a = new ClassA();
        System.out.println("Test ClassA.");
        a.m1();
        a.m2();
        a.m3();
        a.m4();
```

Inheritance

```
ClassB b = new ClassB();
System.out.println("Test ClassB.");
b.m1();
b.m2();
b.m3();
b.m4();

ClassA b0 = new ClassB();
System.out.println("Test_0 ClassB.");
b0.m1();
b0.m2();
b0.m3();
b0.m4();
System.out.println("The End.");  }  }
```

Inheritance

The Start.
Test ClassA.
ClassA, metod m1, i=1
ClassA, metod m2, i=1
ClassA, metod m3, runnind m4(): ClassA, metod m4
ClassA, metod m4
Test ClassB.
ClassB, metod m1, i=1.1
ClassA, metod m2, i=1
ClassA, metod m3, runnind m4(): ClassB, metod m4
ClassB, metod m4
Test_0 ClassB.
ClassB, metod m1, i=1.1
ClassA, metod m2, i=1
ClassA, metod m3, runnind m4(): ClassB, metod m4
ClassB, metod m4
The End.

Java Classes

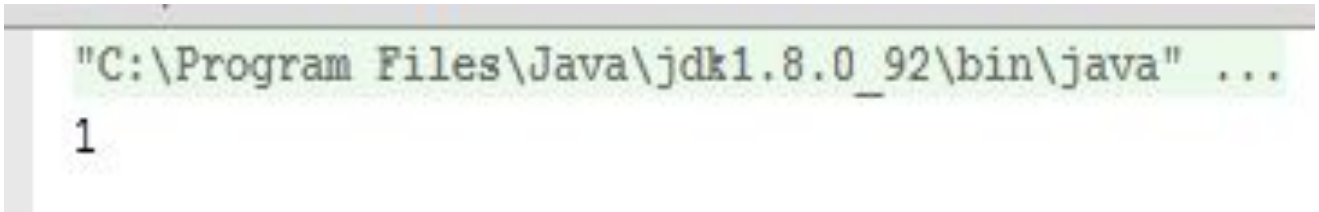
What is **wrong** in the code ?

```
package com.softserve.train;

public class Parent {

    int f( ) {
        return 1;
    }

    public int useF() {
        return f();
    }
}
```

```
package com.softserve.train2;

import
com.softserve.train.Parent;

public class Child extends
Parent {

    int f() {
        return 2;
    }
}
```

Let's check it

```
package com.samples;

import com.softserve.train2.*;

public class OOPSamples {
    public static void main(String... args)
{
        Child child = new Child();
        System.out.println(child.useF());
    }
}
```



Polymorphism

```
public abstract class ACar {
    private double maxSpeed;

    public double getMaxSpeed( ) {
        return maxSpeed;
    }

    public void setMaxSpeed(double maxSpeed) {
        this.maxSpeed = maxSpeed;
    }

    abstract void carRides( );
}
```

Polymorphism

```
public class BmwX6 extends ACar {
    public BmwX6( ) { }

    @Override
    public void carRides( ) {
        setMaxSpeed(200);
        System.out.println("Car Rides");
        workedEngine( );
        workedGearBox( );
    }
    public void workedEngine( ) {
        System.out.println("BmwX6: Engine Running
                        on Petrol.");
        System.out.println("BmwX6: Max Speed: " +
                           getMaxSpeed( ));
    }
```

Polymorphism

```
 private void workedGearBox( ) {
        System.out.println("BmwX6: Worked GearBox.");
 }

 public void lightsShine( ) {
    System.out.println("BmwX6: Halogen Headlights.");
 }
}
```

inheritance of **private** fields and methods ?

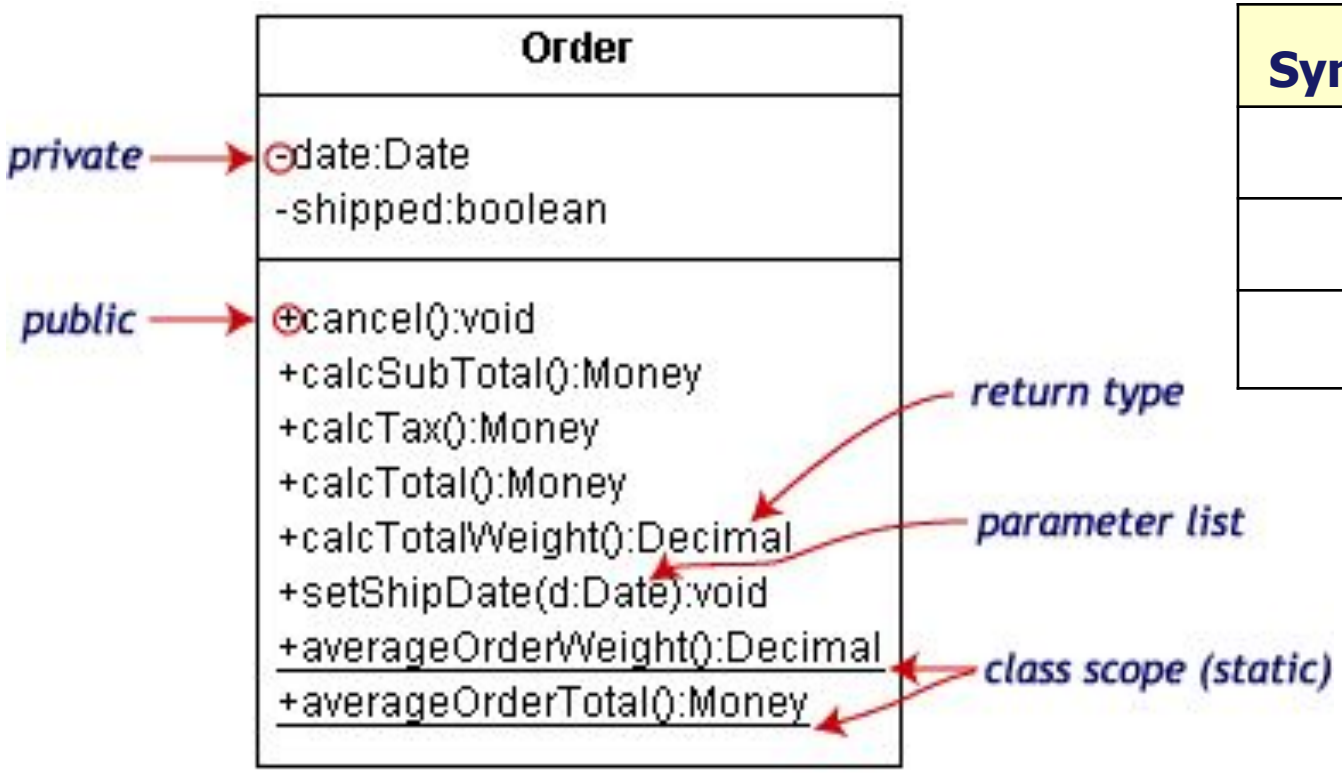
Polymorphism

```
public class BmwX6mod extends BmwX6 {
    public BmwX6mod( ) {
        super( );
    }
   @Override
    public void workedEngine( ) {
        System.out.println("BmwX6mod: Engine
                    Running on Diesel.");
        System.out.println("BmwX6mod: Max Speed: " +
                                        getMaxSpeed( ));
    }
   @Override
    public void lightsShine( ) {
        System.out.println("BmwX6mod: Xenon Headlights.");
        super.lightsShine();
    }
}
```

Polymorphism

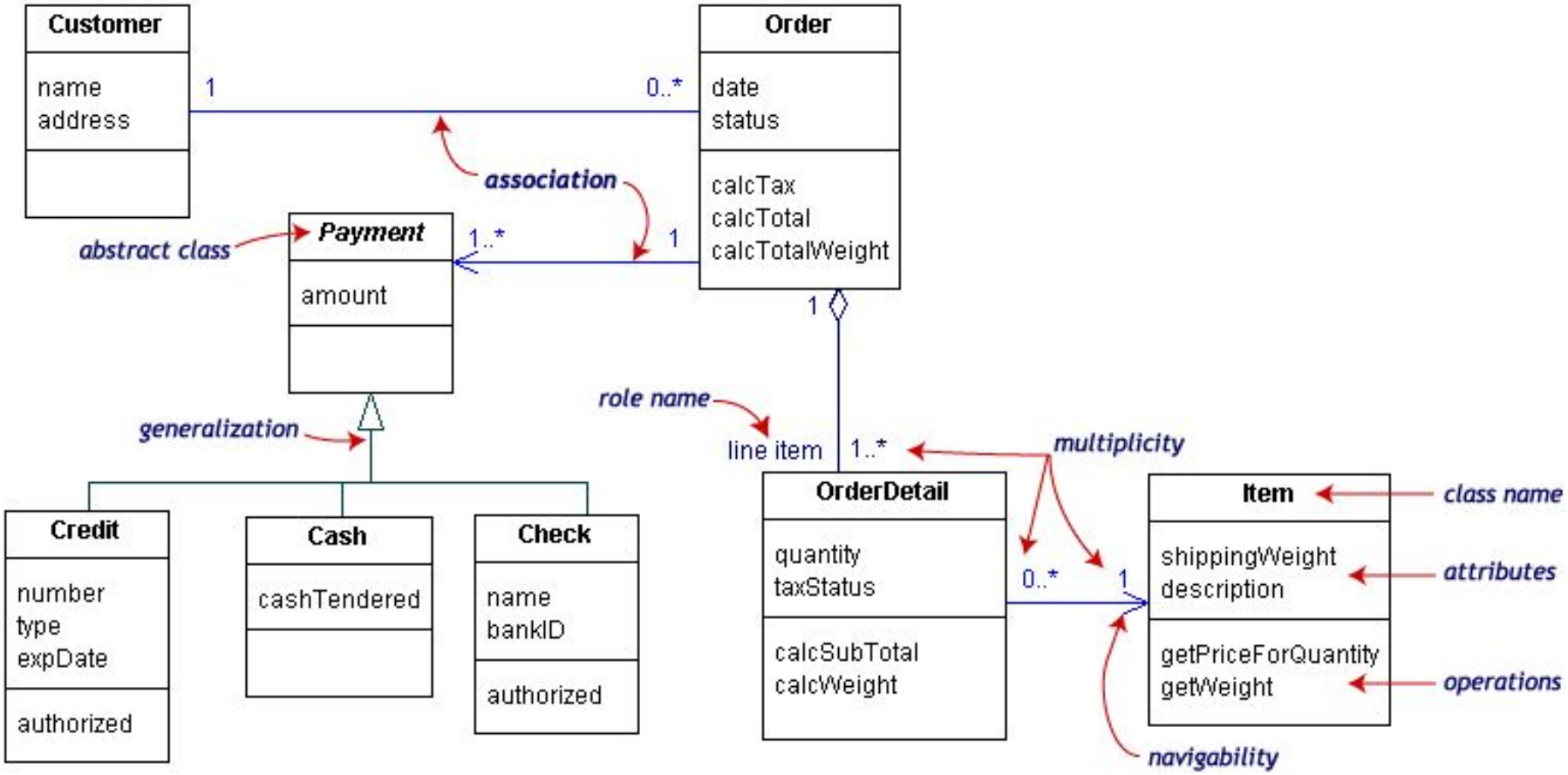
```
public class Appl {
    public static void main(String[ ] args) {
        ACar carX6 = new BmwX6( );
      carX6.carRides( );
        ((BmwX6)carX6).lightsShine( );

      ACar carX6mod = new BmwX6mod( );
        carX6mod.carRides( );
        ((BmwX6)carX6mod).lightsShine( );

      BmwX6 carX6mod2 = new BmwX6mod( );
        carX6mod2.carRides( );
        carX6mod2.lightsShine( );
    }
}
```

Class Diagram. Visibility and scope



Symbol	Access
+	public
-	private
#	protected

Class Diagram



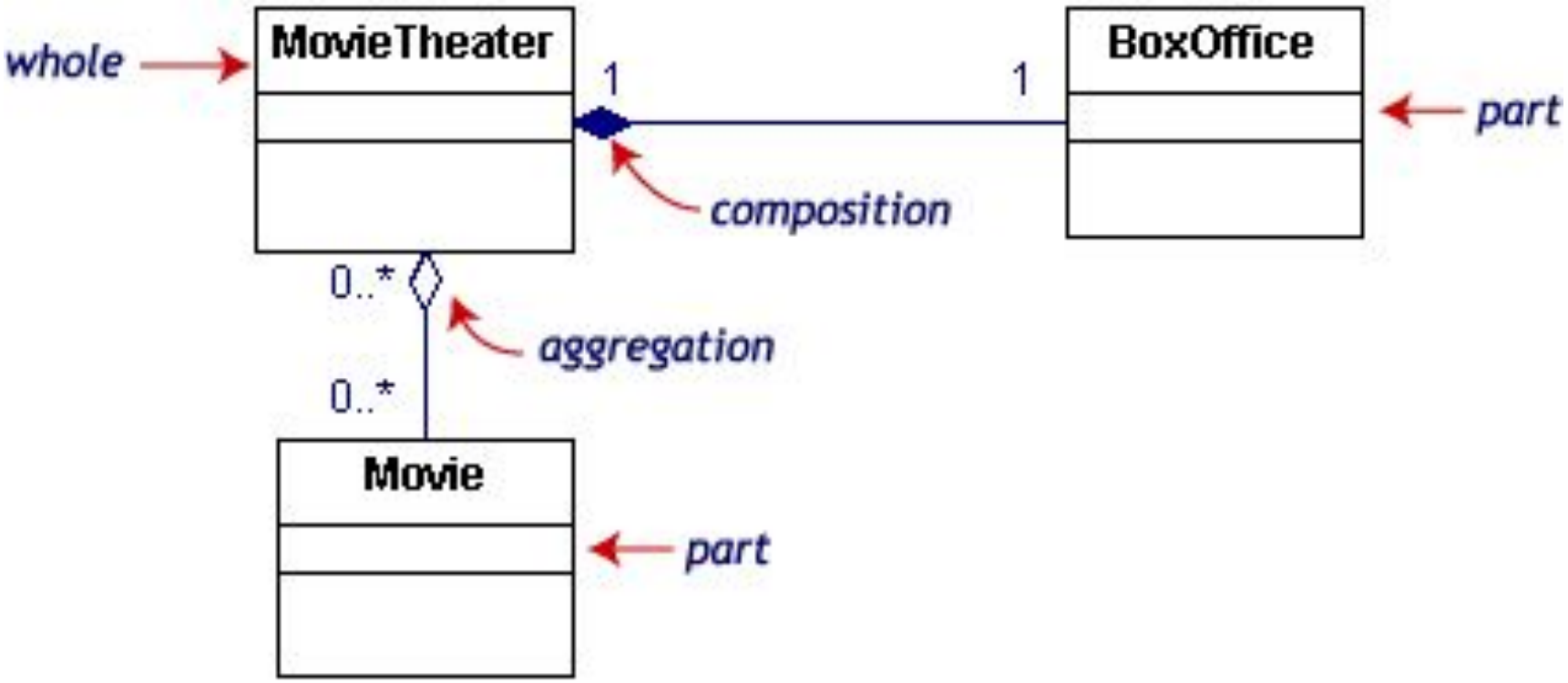
Class Diagram

- Our class diagram has three kinds of relationships.
  - **association** -- a relationship between instances of the two classes. There is an association between two classes if an instance of one class must know about the other in order to perform its work. In a diagram, an association is a link connecting two classes.
  - **aggregation** -- an association in which one class belongs to a collection. An aggregation has a diamond end pointing to the part containing the whole. In our diagram, Order has a collection of OrderDetails.
  - **generalization** -- an inheritance link indicating one class is a superclass of the other. A generalization has a triangle pointing to the superclass. Payment is a superclass of Cash, Check, and Credit.

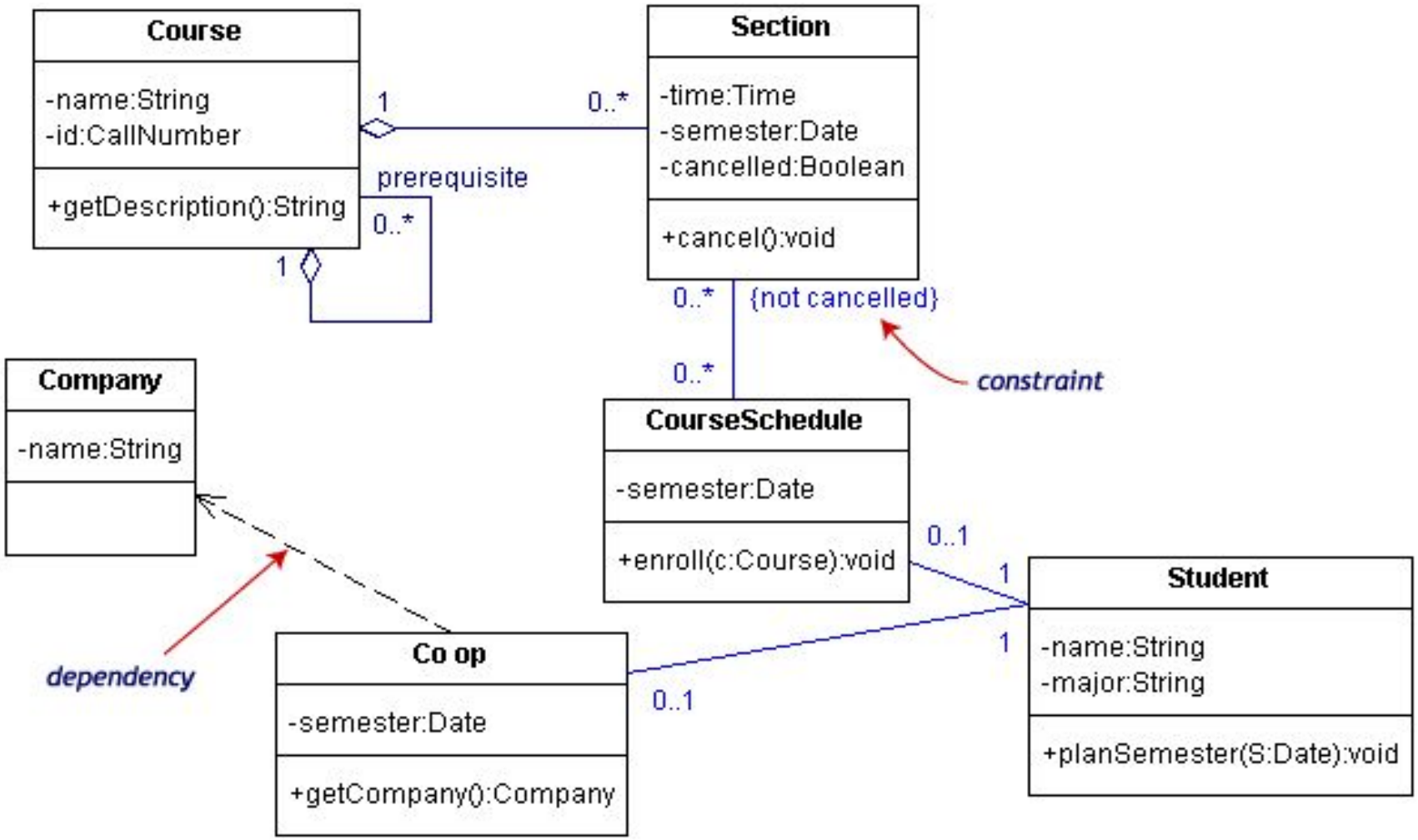
Class Diagram. Multiplicities

Multiplicities	Meaning
0..1	zero or one instance. The notation n . . M indicates n to m instances.
0..* or *	no limit on the number of instances (including none).
1	Exactly one instance
1..*	at least one instance

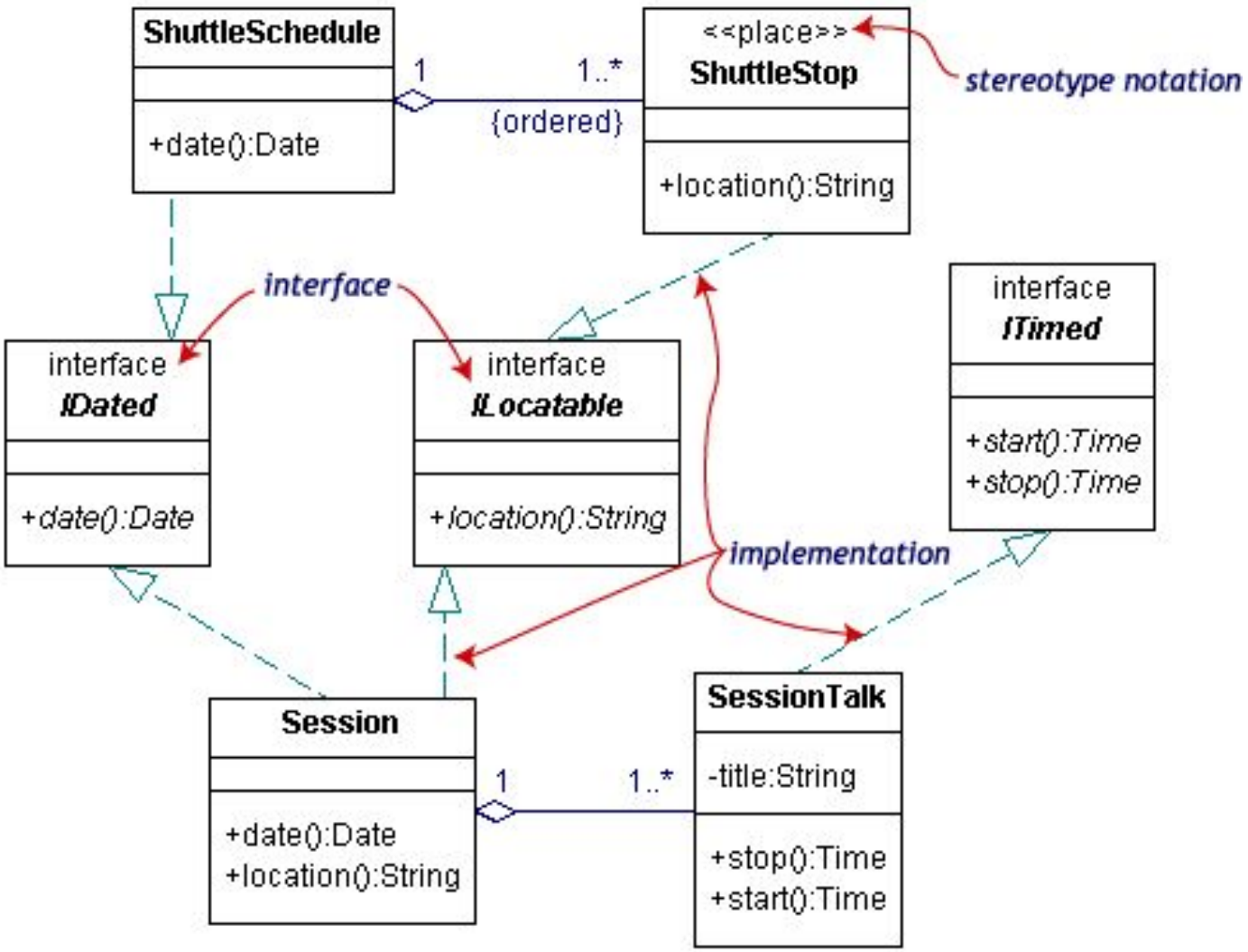
Composition and aggregation



Dependencies and constraints



Class Diagram. Interfaces and stereotypes



final

- A ***final variable*** can only be assigned once and its value cannot be modified once assigned.

Constants are variables defined

```
final double RADIUS = 10;
```

- A ***final method*** cannot be overridden by subclasses

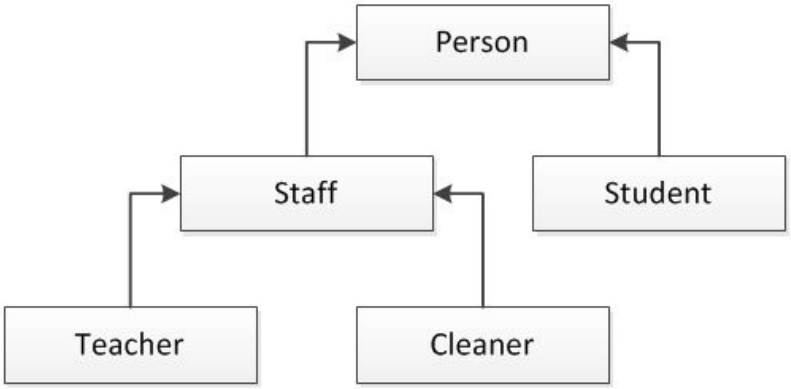
```
public final void myFinalMethod() {...}
```

- A ***final class*** cannot extend

```
public final class MyFinalClass {...}
```

Practical tasks

1. Create interface ***Animal*** with methods ***voice()*** and ***feed()***. Create two classes ***Cat*** and ***Dog***, which implement this interface. Create array of *Animal* and add some Cats and Dogs to it. Call *voice()* and *feed()* method for all of it
2. Create next structure. In abstract class ***Person*** with property ***name***, declare abstract method ***print***(). In other classes in body of method *print*() output text "I am a …". In class Staff declare abstract method ***salary***(). In each concrete class create constant *TYPE_PERSON*. Output type of person in each constructors. Create array of *Person* and add some Teachers, Cleaners and Students to it. Call method *print*() for all of it. Call method *salary*() for all Teachers and Cleaner



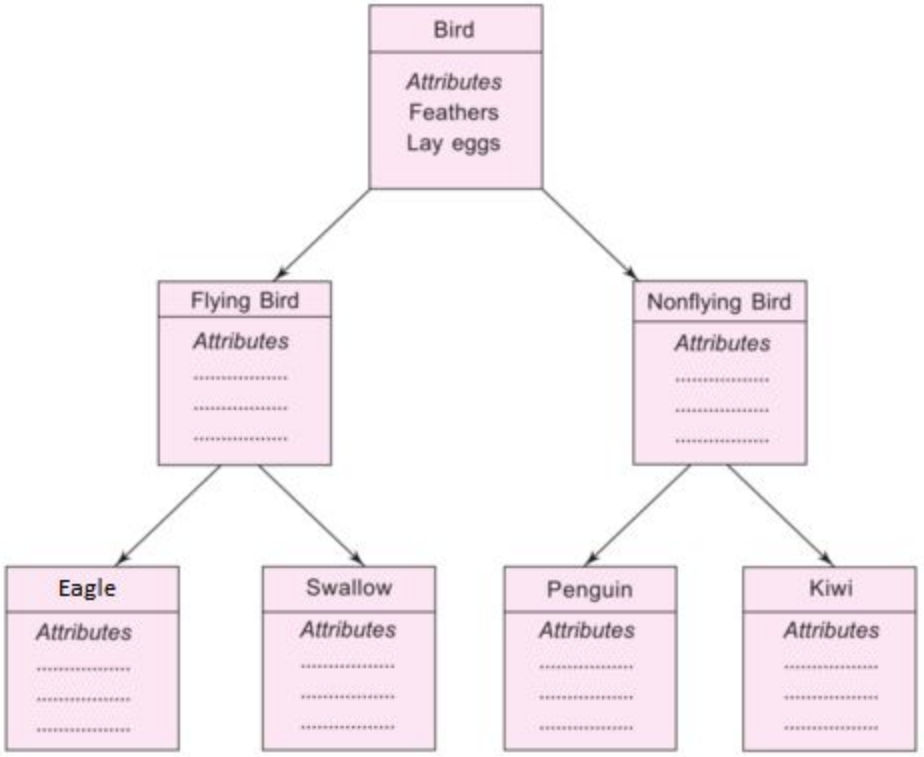
HomeWork (online course)

- UDEMY course "Java Tutorial for Complete Beginners": https://www.udemy.com/java-tutorial/
- Complete lessons 26-31:

Homework

1. Develop abstract class *Bird* with attributes *feathers* and *layEggs* and an abstarct method *fly*(). Develop classes *FlyingBird* and *NonFlyingBird*. Create class Eagle, Swallow, Penguin and Chicken.

Create array *Bird* and add different birds to it. Call *fly()* method for all of it. Output the information about each type of created bird.



Homework

2. Create an interface to the method *calculatePay()*, the base class Employee with a string variable *employeeId*. Create two classes *SalariedEmployee* and *ContractEmployee*, which implement interface and are inherited from the base class.
   - Describe hourly paid workers in the relevant classes (one of the children), and fixed paid workers (second child).
   - Describe the string variable *socialSecurityNumber* in the class SalariedEmployee .
   - Include a description of *federalTaxIdmember* in the class of contractEmployee .
   - The calculation formula for the "time-worker" is: "*the average monthly salary = hourly rate * number of hours worked*"

Homework

- For employees with a fixed payment the formula is: "*the average monthly salary = fixed monthly payment*"
- Create an array of employees and add the employees with different form of payment.
- Arrange the entire sequence of workers descending the average monthly wage. Output the employee *ID*, *name*, and the average monthly wage for all elements of the list.

softserve experience matters

The end

USA HQ
Toll Free: 866-687-3588
Tel: +1-512-516-8880

Ukraine HQ
Tel: +380-32-240-9090

Bulgaria
Tel: +359-2-902-3760