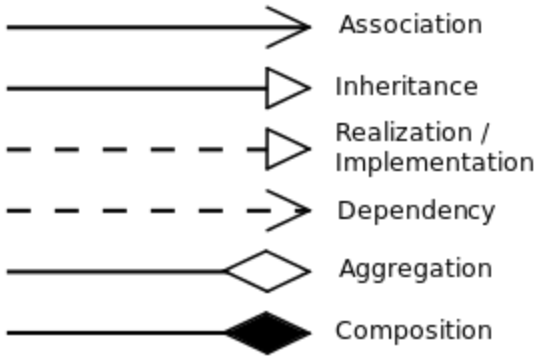
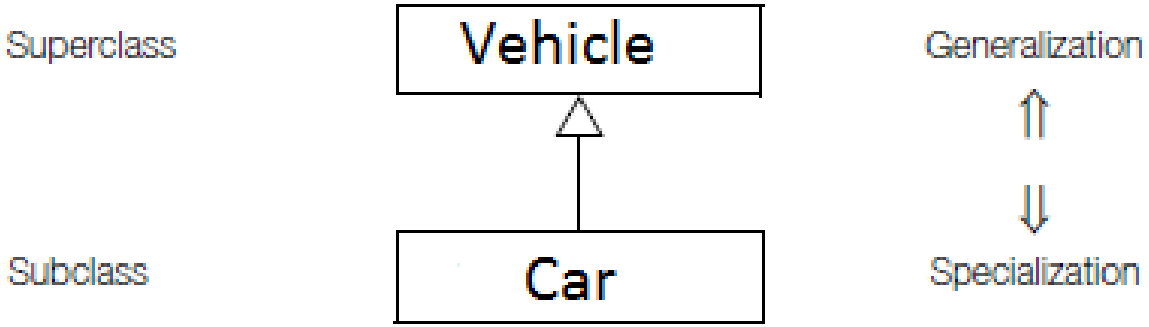
OOP LAB 1

Marin Iuliana
marin.iulliana25@gmail.com

PL REVISION



- inheritance
- association
- difference between **Association**, **Aggregation (wallet and money)** and **Composition (human and heart)**
- hiding, overloading, overriding



HIDING A STATIC METHOD AND OVERRIDING AN INSTANCE METHOD

```
public class Animal {
    public static void testClassMethod() {
            System.out.println("The static method in Animal");
    }
    public void testInstanceMethod() {
            System.out.println("The instance method in Animal");
    }
}
```

AND

```
public class Cat extends Animal {
    public static void testClassMethod() {
            System.out.println("The static method in Cat");
    }
    public void testInstanceMethod() {
            System.out.println("The instance method in Cat");
    }
    public static void main(String[] args) {
            Cat myCat = new Cat();
            Animal myAnimal = myCat;
            Animal.testClassMethod();
            myAnimal.testInstanceMethod();
    }
}
```

OUTPUT:
The static method in Animal
The instance method in Cat

POLYMORPHISM

- Static

Overloading -> Return type of method does not matter; Argument list should be different

Done at compile time

The content is different

- Dynamic

Done during runtime

Overriding => covariant return; Argument list should be the same;

- Static methods can be overloaded => a class can have more than one static method of same name
- private and final methods can be overloaded but they cannot be overridden

EXAMPLE

```
class C1{
   public MyType1 m(){ }
}
class C2{
   public MyType2 m(){}
}
```

We can override if MyType2 is a subclass of MyType1.

A subclass within the same package as the instance's superclass can override any superclass method that is not private or final.

CAN I OVERRIDE A STATIC METHOD?

Answer: NEVER!!!

```
class Foo {
    public static void method() {
        System.out.println("in Foo");
    }
}

class Bar extends Foo {
    public static void method() {
        System.out.println("in Bar");
    }
}
```

Example of a static method *hiding* another static method

- When you *override* a method, you still get the benefits of run-time polymorphism, and when you *hide*, you don't.

CODE SAMPLE

```
class Foo {
    public static void classMethod() {
        System.out.println("classMethod() in Foo");
    }

    public void instanceMethod() {
        System.out.println("instanceMethod() in Foo");
    }
}

class Bar extends Foo {
    public static void classMethod() {
        System.out.println("classMethod() in Bar");
    }

    public void instanceMethod() {
        System.out.println("instanceMethod() in Bar");
    }
}

class Test {
    public static void main(String[] args) {
        Foo f = new Bar();
        f.instanceMethod();
        f.classMethod();
    }
}
```

one is overriding and the other is hiding

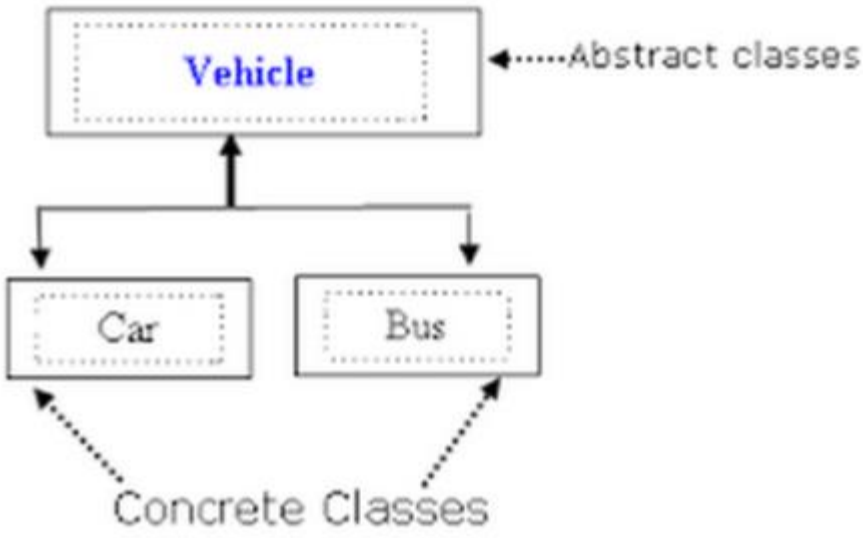
- If you run this, the output is

instanceMethod() in Bar

classMethod() in Foo

ABSTRACT CLASSES

- may or may not include **abstract** methods
- cannot be instantiated
- they can be subclassed
- **abstract** method is a method that is declared without an implementation (without braces, and followed by a semicolon)



ABSTRACT CLASS EXAMPLE

```
public abstract class Person {
    private String name;
    private String gender;
    public Person(String nm, String gen){
            this.name=nm;
            this.gender=gen;
    }
    //abstract method
    public abstract void work();
    @Override
    public String toString(){
            return "Name="+this.name+"::Gender="+this.gender;
    }
    public void changeName(String newName) {
            this.name = newName;
    }
}
```

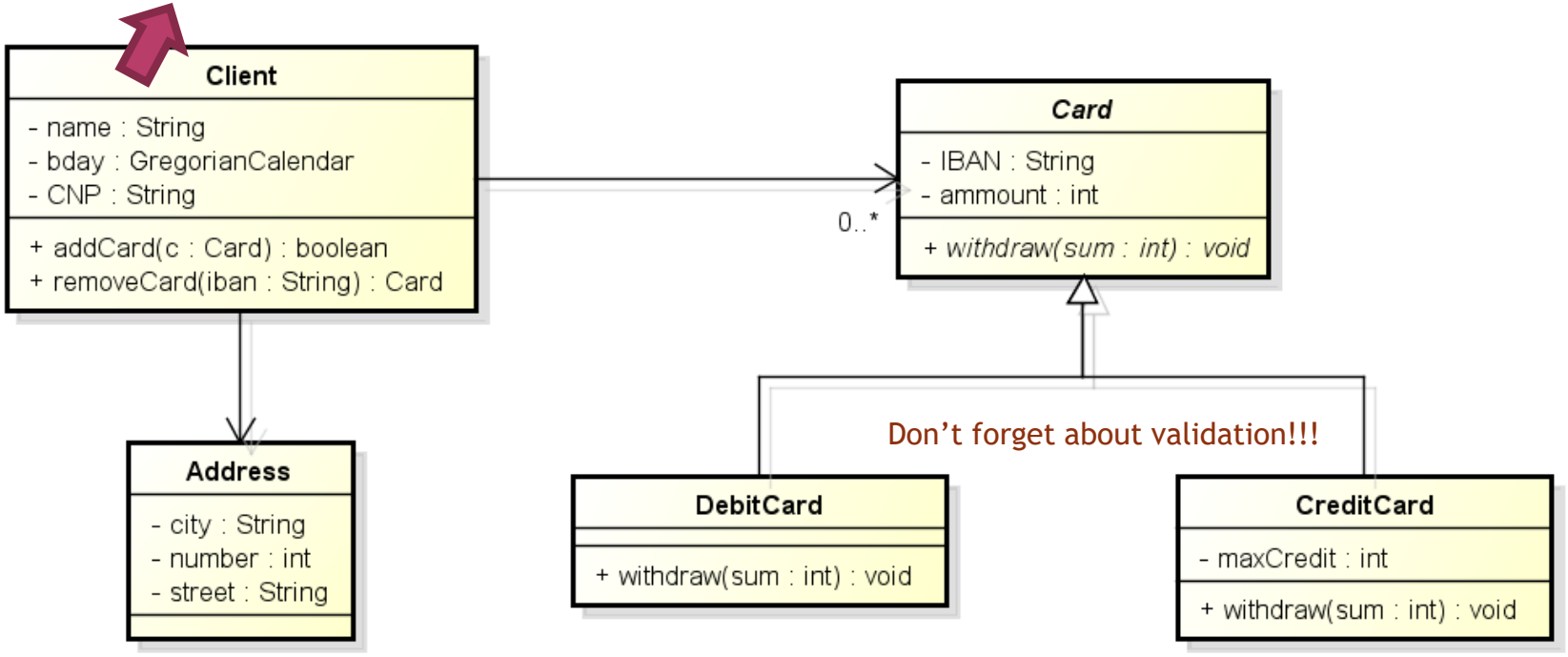
ABSTRACT CLASS EXAMPLE

```
public class Employee extends Person {
    private int empId;
    public Employee(String nm, String gen, int id) {
            super(nm, gen);
            this.empId=id;
    }
    @Override
    public void work() {
            if(empId == 0){
                    System.out.println("Not working");
            }else{
                    System.out.println("Working as employee!!");
            }
    }
    public static void main(String args[]){
            //coding in terms of abstract classes
            Person student = new Employee("Dorina","Female",0);
            Person employee = new Employee("Paul","Male",123);
            student.work();
            employee.work();
            //using method implemented in abstract class - inheritance
            employee.changeName("Pankaj Kumar");
            System.out.println(employee.toString());
    }
}
```

PROBLEM 1

-> To automatically have getters and setters
Right Click + “Insert Code…” + Option

```
private List<Card> cards = new ArrayList<>();
```



To go through the list of cards:

```
for (Iterator<Card> it = cards.iterator(); it.hasNext();) {
        if (it.next().getIBAN().equals(IBAN)) {
            it.remove(); …
```

OR `for(Card c: cards){ if(c instanceof CreditCard){ …`

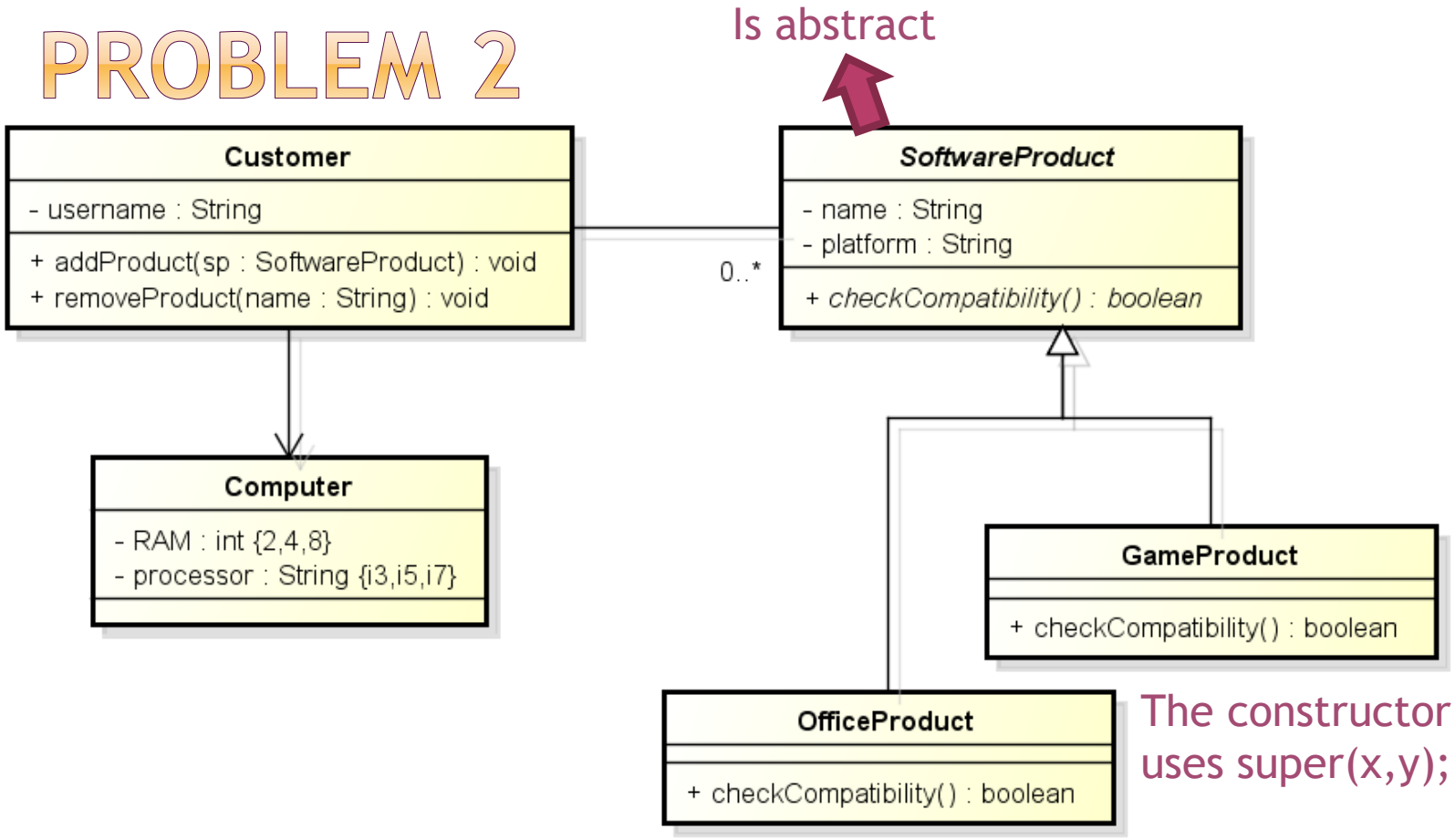
GREGORIAN CALENDAR

- SimpleDateFormat sdf = new SimpleDateFormat("yyyy MMM dd");
- Calendar calendar = new GregorianCalendar(2019,8,25);
- System.out.println("Date : " + sdf.format(calendar.getTime()));

Which is the month's name?

Is it August or September?

PROBLEM 2



- The RAM and processor should receive values taken from the given set.
- Checking the constraints is done in the constructor with throw Exception

Like:

```
public Computer(int ram, String processor) throws Exception {
        if(ram!=2 && ram!=4 && ram!=8) {
            throw new Exception("Bad value for RAM");
        }…
```