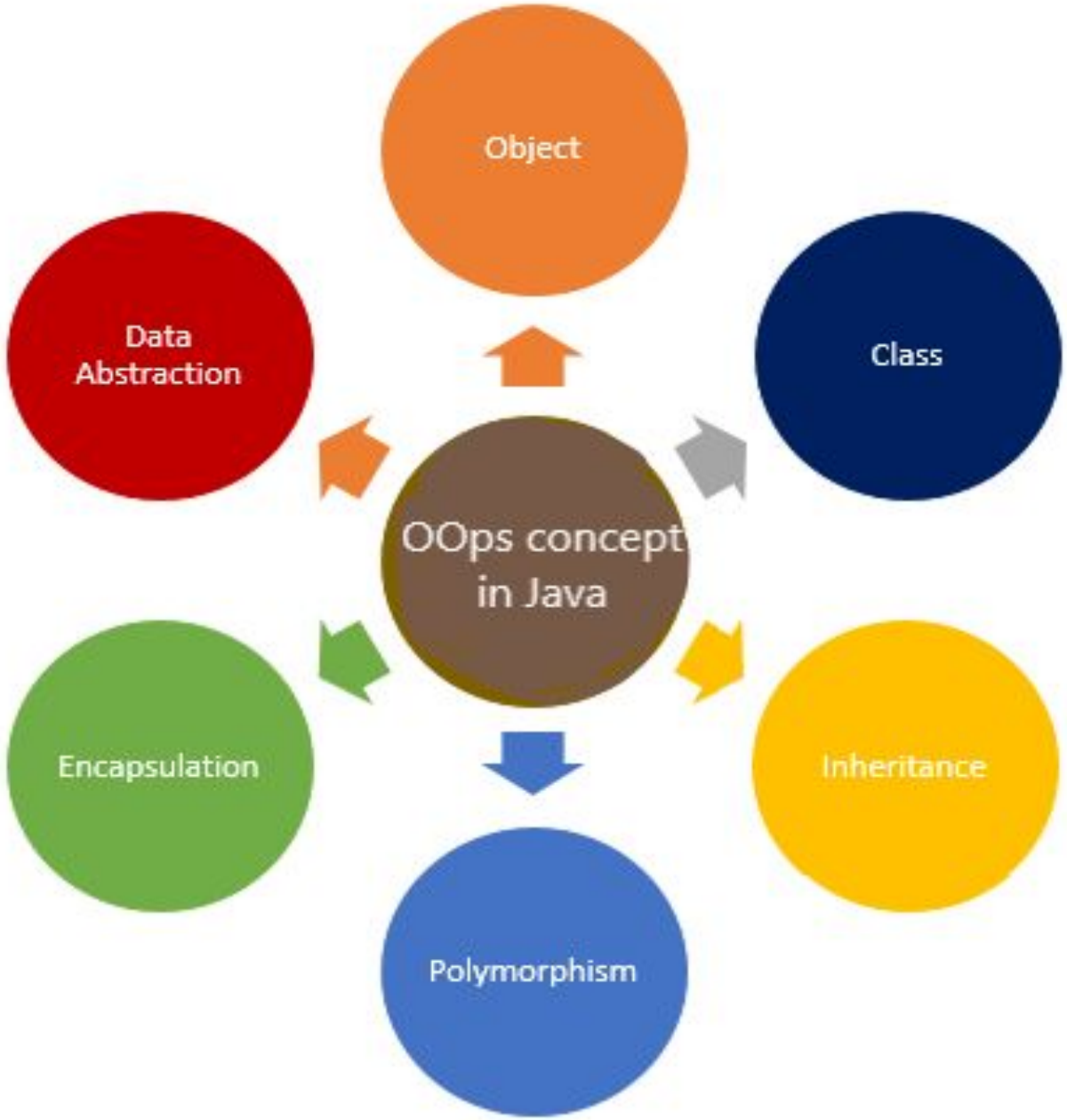
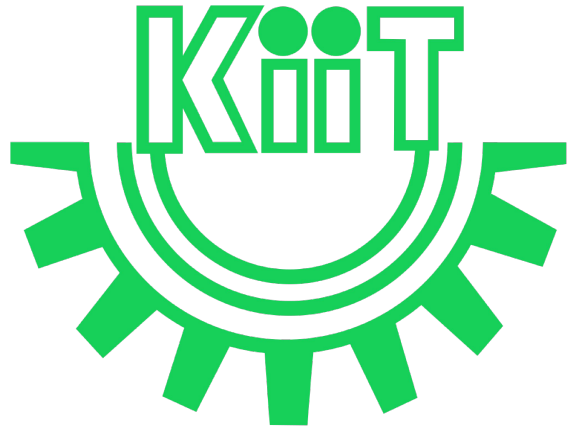


CS20004: Object Oriented Programming using Java

Lec-13

In this Discussion . . .

- Polymorphism
  - Compile time
  - Runtime
    - Dynamic method dispatch
- References



Polymorphism

- Polymorphism is one of the main aspects of Object-Oriented Programming (OOP).
- Polymorphism in Java is a concept by which we can perform a *single action in different ways*.
- Polymorphism is derived from 2 Greek words: “poly” and “morphs”. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

Polymorphism Examples

- Polymorphism through the lens of an example;
  - A lady can have different characteristics simultaneously. She can be a mother, a daughter, or a wife, so the same lady possesses different behavior in different situations.
  - Another example of polymorphism can be seen in carbon, as carbon can exist in many forms, i.e., diamond, graphite, coal, etc.
- We can say that both woman and carbon show different characteristics at the same time according to the situation. This is called **polymorphism**.
- The definition of polymorphism can be explained as performing a single task in different ways. A single interface having multiple implementations is also called polymorphism.

How can polymorphism be achieved in Java?

- Polymorphism in Java can be achieved in two ways:
  - method overloading and
  - method overriding.
- Polymorphism in Java is mainly divided into two types:
  - Compile-time polymorphism
  - Runtime polymorphism
- Compile-time polymorphism can be achieved by method overloading, and Runtime polymorphism can be achieved by method overriding.

Polymorphism

```
class Parent {
  // creating print method
  void print() {System.out.println("Hi I am parent");}
}
class Child extends Parent {
  // overriding print method
  void print() {System.out.println("Hi I am children");}}
//Class to illustrate compile-time polymorphism
class Overload {
  // Creating a statement method
  void statement(String name) {System.out.println("Hi myself "
+ name);}
  // overloading statement method
  void statement(String fname, String lname) {
    System.out.println("Hi myself " + fname + " " + lname);}}
public class Testover {
  public static void main(String[] args) {
    // creating instance of parent
    Parent obj1;obj1 = new Parent(); obj1.print();
    obj1 = new Child();obj1.print();
    // creating instance of overload
    Overload obj2 = new Overload();
    obj2.statement("Sourajit.");
    obj2.statement("Sourajit", "Behera.");
  }}
```

In the example, the Child class extends the Parent class, and the print method is overridden, this represents run-time polymorphism in Java.

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Polymorphism$ javac Te
stover.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Polymorphism$ java Tes
tover
Hi I am parent
Hi I am children
Hi myself Sourajit.
Hi myself Sourajit Behera.
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Polymorphism$
```

In the Overload class, the statement function is overloaded, this represents compile-time polymorphism in Java.

Polymorphism in real-life

- We can relate polymorphism in real life by the following example. Consider a parent class as living thing.
- Living things exist on the planet in the forms of human beings, animals, plants, bacteria, etc. These are the child classes inherited from the parent class, i.e., living things.

Polymorphism in real-life

```
class Livingthings {
  public void live() {
    System.out.print("live on ");
  }
}
class Animals extends Livingthings {
  public void live() {
    System.out.println("water, air, and land.");
  }
}
class Humanbeing extends Livingthings {
  public void live() {
    System.out.println("land.");
  }
}
class Polymorphismexp {
  public static void main(String[] args) {
    Livingthings LT = new Livingthings();
    Livingthings LT1;
    LT1 = new Animals();
    System.out.print("Animals ");
    LT.live();
    LT1.live();
    LT1 = new Humanbeing();
    System.out.print("Human beings ");
    LT.live();
    LT1.live();
  }}
```

- In this example, the parent class Livingthings has the method live(). Subclasses of Livingthings i.e., Animals, and Humanbeing have their way of living.

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Polymorphism$ javac Polymorphismexp.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Polymorphism$ java Polymorphismexp
Animals live on water, air, and land.
Human beings live on land.
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Polymorphism$
```

- By the principle of inheritance and polymorphism, each subclass can define its way of living with the help of the live() method.

Types of Polymorphism in Java

- There are two main types of polymorphism in Java.
  - Compile-time polymorphism
    - This type of polymorphism in Java is also called static polymorphism or static method dispatch.
    - It can be achieved by method overloading.
    - In this process, an overloaded method is resolved at compile time rather than resolving at runtime.

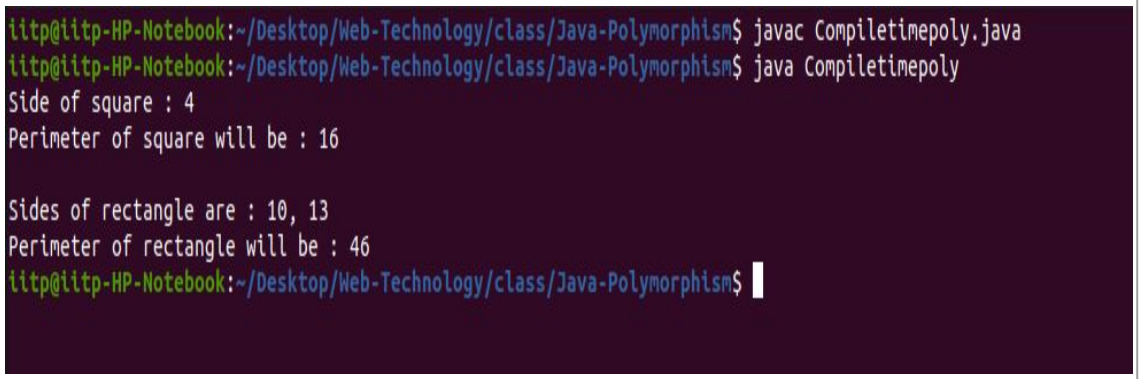
Types of Polymorphism in Java

- There are two main types of polymorphism in Java.
  - Compile-time polymorphism
    - Method overloading
      - Consider a class where multiple methods have the same name. It will be difficult for the compiler to distinguish between every method.
      - To overcome this problem, we pass a different number of arguments to the method or different types of arguments to the method. In this way, we achieve method overloading.
      - In other words, a class can have multiple methods of the same name, and each method can be differentiated either by passing different types of parameters or by passing a different number of parameters.

Types of Polymorphism in Java

- Compile-time polymorphism
  - Method overloading (Passing different numbers of arguments to the function.)

```
class D {
  // perimeter method with a single argument
  static int perimeter(int a) {
    return 4 * a;
  }
  // perimeter method with two arguments (overloading)
  static int perimeter(int l, int b) {
    return 2 * (l + b);
  }
}
class Compiletimepoly {
  public static void main(String[] args) {
    // calling perimeter method by passing a single argument
    System.out.println("Side of square : 4\nPerimeter of square will be
: " + D.perimeter(4) + "\n");
    // calling perimeter method by passing two arguments
    System.out.println("Sides of rectangle are : 10, 13\nPerimeter of
rectangle will be : " + D.perimeter(10, 13));
  }
}
```

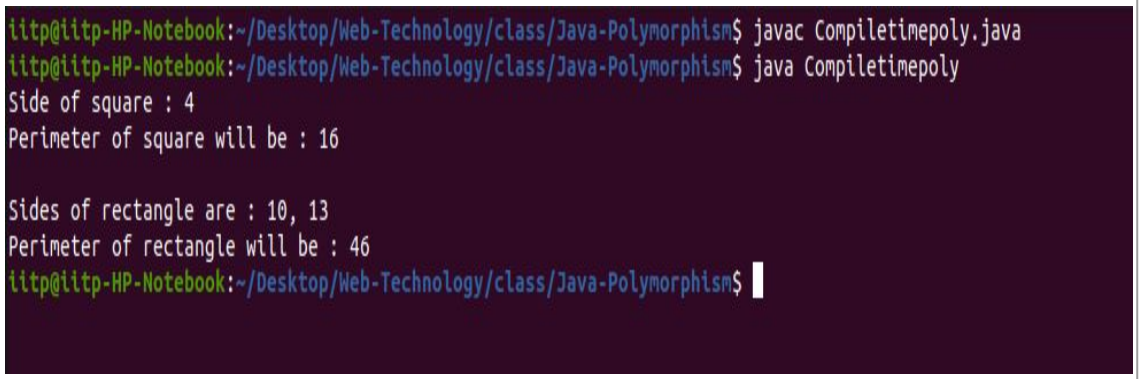


Types of Polymorphism in Java

- Compile-time polymorphism
  - Method overloading (Passing different numbers of arguments to the function.)

```
class D {
  // perimeter method with a single argument
  static int perimeter(int a) {
    return 4 * a;
  }
  // perimeter method with two arguments (overloading)
  static int perimeter(int l, int b) {
    return 2 * (l + b);
  }
}
class Compiletimepoly {
  public static void main(String[] args) {
    // calling perimeter method by passing a single argument
    System.out.println("Side of square : 4\nPerimeter of square will be
: " + D.perimeter(4) + "\n");
    // calling perimeter method by passing two arguments
    System.out.println("Sides of rectangle are : 10, 13\nPerimeter of
rectangle will be : " + D.perimeter(10, 13));
  }
}
```

- In the example, the class D has two functions, both having the same name, but the first function has a single argument to pass, and another one has two arguments to pass.



Here, perimeter() function calculates the perimeter of square & rectangle. This way, two functions having the same name are distinguished, and compile-time polymorphism achieved.

Types of Polymorphism in Java

- Compile-time polymorphism
  - Method overloading (Passing different types of argument to the function.)

```
class K{
   // contact method, which takes two arguments String and long
        static void contact(String fname, long number) {
                System.out.println("Name : "+fname+"\nNumber :
"+number);
        }
   // contact method, which takes two arguments and both are
Strings (overloading)
        static void contact(String fname, String mailid) {
                System.out.println("Name : "+fname+"\nEmail : "+mailid);
        }}
class Compilepoly{
   public static void main(String[] args) {
      // calling first contact method
      K.contact("Sourajit", 1234567890);
      System.out.print("\n");
      // calling second contact method
      K.contact("Sourajit", "sourajit.beherafcs@kiit.ac.in");
   }}
```

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Polymorphism$ javac Compilepoly.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Polymorphism$ java Compilepoly
Name : Sourajit
Number : 1234567890

Name : Sourajit
Email : sourajit.beherafcs@kiit.ac.in
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Polymorphism$
```

Types of Polymorphism in Java

- Compile-time polymorphism
  - Method overloading (Passing different types of argument to the function.)

```
class K{
   // contact method, which takes two arguments String and long
        static void contact(String fname, long number) {
                System.out.println("Name : "+fname+"\nNumber :
"+number);
        }
   // contact method, which takes two arguments and both are
Strings (overloading)
        static void contact(String fname, String mailid) {
                System.out.println("Name : "+fname+"\nEmail : "+mailid);
        }}
class Compilepoly{
   public static void main(String[] args) {
      // calling first contact method
      K.contact("Sourajit", 1234567890);
      System.out.print("\n");
      // calling second contact method
      K.contact("Sourajit", "sourajit.beherafcs@kiit.ac.in");
   }}
```

- In the example, class K has two functions, both having the same name, but in the first function, we pass a string and long as an argument, and in the second function, we pass two strings.

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Polymorphism$ javac Compilepoly.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Polymorphism$ java Compilepoly
Name : Sourajit
Number : 1234567890

Name : Sourajit
Email : sourajit.beherafcs@kiit.ac.in
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Polymorphism$
```

It shows that we can save the person's contact by mobile number or email. In this way, compile-time polymorphism is achieved.

Types of Polymorphism in Java

- There are two main types of polymorphism in Java.
  - Runtime polymorphism
    - Runtime polymorphism is also called Dynamic method dispatch.
    - Instead of resolving the overridden method at compile-time, it is resolved at runtime.
    - Here, an overridden child class method is called through its parent's reference. Then the method is evoked according to the type of object.

Types of Polymorphism in Java

- There are two main types of polymorphism in Java.
  - Runtime polymorphism
    - In runtime, JVM figures out the object type and the method belonging to that object.
    - Runtime polymorphism in Java occurs when we have two or more classes, and all are interrelated through inheritance.
    - To achieve runtime polymorphism, we must build an "IS-A" (Inheritance) relationship between classes and override a method.

Types of Polymorphism in Java

- There are two main types of polymorphism in Java.
  - Runtime polymorphism
    - Method overriding
      - If a child class has a method as its parent class, it is called method overriding.
      - If the derived class has a specific implementation of the method that has been declared in its parent class is known as method overriding.

Types of Polymorphism in Java

- Runtime polymorphism

```
class Shape{
        void area(){
                System.out.println("Formula for areas.");
        }}
class Square extends Shape{
        void area(){
                System.out.println("Area of square : a * a");
        }}
class Rectangle extends Shape{
        void area(){
                System.out.println("Area of rectangle : 2 * (a + b)");
        }}
class Runtimepoly{
   public static void main(String[] args) {
        Shape S = new Shape();
        S.area();
        S = new Square();
        S.area();
        S = new Rectangle();
        S.area();
        }}
```

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Polymorphism$ javac Runtimepoly.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Polymorphism$ java Runtimepoly
Formula for areas.
Area of square : a * a
Area of rectangle : 2 * (a + b)
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Polymorphism$
```

Rule: Runtime polymorphism can't be achieved by data members.

Java Runtime Polymorphism with Data Member

- A method is overridden, not the data members, so runtime polymorphism can't be achieved by data members.

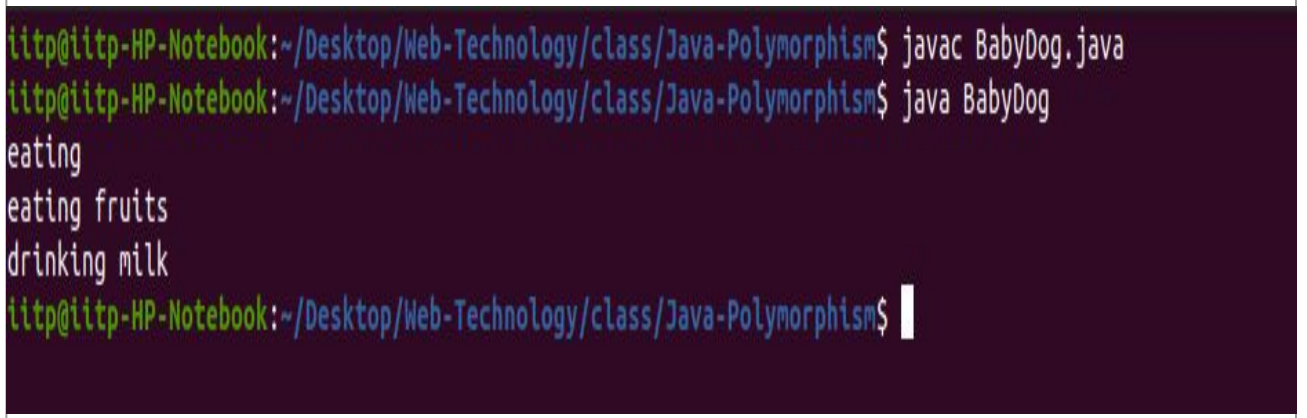
```
class Bike{
 int speedlimit=90;
}
class Honda3 extends Bike{
 int speedlimit=150;

 public static void main(String args[]){
  Bike obj=new Honda3();
  System.out.println(obj.speedlimit);//90
}
```

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Polymorphism$ javac Honda3.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Polymorphism$ java Honda3
90
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Polymorphism$
```

In the example, both the classes have a data member speedlimit. We are accessing the data member by the reference variable of Parent class which refers to the subclass object. Since we are accessing the data member which is not overridden, hence it will access the data member of the Parent class always.

Java Runtime Polymorphism with Multilevel Inheritance

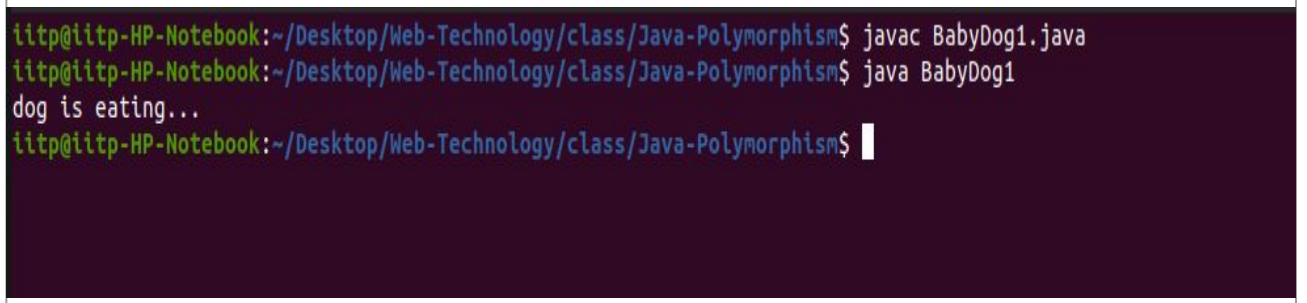
```
class Animal{
void eat(){System.out.println("eating");}
}
class Dog extends Animal{
void eat(){System.out.println("eating fruits");}
}
class BabyDog extends Dog{
void eat(){System.out.println("drinking milk");}
public static void main(String args[]){
Animal a1,a2,a3;
a1=new Animal();
a2=new Dog();
a3=new BabyDog();
a1.eat();
a2.eat();
a3.eat();
}
}
```



Java Runtime Polymorphism with Multilevel Inheritance

- Since, BabyDog is not overriding the eat() method, so eat() method of Dog class is invoked.

```
class Animal{
void eat(){System.out.println("animal is eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("dog is eating...");}
}
class BabyDog1 extends Dog{
public static void main(String args[]){
Animal a=new BabyDog1();
a.eat();
}}
```



References

1. https://www.scaler.com/topics/java/polymorphism-in-java/
2. https://www.tutorialspoint.com/java/java_polymorphism.htm
3. https://www.javatpoint.com/runtime-polymorphism-in-java
4.
5.
6.