Inheritance and polymorphism

Adam Krechowicz

Inheritance

Inheritance

Inheritance is a process of creating a hierarchy of classes is such a way that a subclass contains all the fields and methods from the superclass.

- relation "is a"
- Dog is an animal
- Car is a vehicle
- Vehicle is an object

Inheritance in java

```
1  class Base{
2      void baseMethod(){
3          System.out.println("Base method");
4      }
5  }
6
7  public class Inherit extends Base{
8      void method(){
9          System.out.println("Method");
10     }
11
12     public static void main(String[] args){
13         Inherit i = new Inherit();
14         i.method();
15         i.baseMethod();
16     }
17 }
```

Properties of inheritance

- Creating the structure of classes
- Reducing repetition of code
- Uniform access to all classes
- Ease to add new features

Multiple inheritance

Multiple inheritance

In java we can only inherit from one class

```
1  class BaseMulti{
2      int pole;
3  }
4
5  class OtherBaseMulti{
6      int pole;
7  }
8
9  //public class Multi extends BaseMulti, OtherBaseMulti {}
```

- Multi inheritance may cause name conflicts
- The only way to deal with multi inheritance is to use interfaces

Multi inheritance

```
1  public class Multiple extends Inherit {
2      void method2(){
3          System.out.println("Other method");
4      }
5
6      public static void main(String[] args){
7          Multiple m = new Multiple();
8          m.baseMethod();
9          m.method();
10         m.method2();
11     }
12
13 }
```

Class that do not inherit any other class

```
1  public class NoExtends{
2
3      public static void main(String[] args){
4          NoExtends ne = new NoExtends();
5          System.out.println(ne.toString());
6          System.out.println(ne.hashCode());
7          System.out.println(ne.getClass());
8      }
9
10 }
```

Class that do no inherit any other class

```
1  public class NoExtends extends Object {
2
3      public static void main(String[] args){
4          NoExtends ne = new NoExtends();
5          System.out.println(ne.toString());
6          System.out.println(ne.hashCode());
7          System.out.println(ne.getClass());
8      }
9
10 }
```

Casting

```
1  class Bazowa{ void m(){System.out.println("base");}}
2  class Pochodna extends Bazowa{void n(){System.out.println("derived");}}
3
4  public class Cast extends Pochodna { void o(){System.out.println("derived derived")
       ;}
5      public static void main(String[] args){
6          Cast c = new Cast();
7          Bazowa b = c;
8          b.m(); //b.n(); b.o();
9          Pochodna p = c;
10         p.m(); p.n(); //p.o();
11         //Cast c1 = b;
12         //Pochodna p1 = b;
13         Cast c2 = new Cast();
14         c2.m(); c2.n(); c2.o();
15         Bazowa b2 = c2;
16         b2.m();  //b2.n(); b2.o();
17         Cast c3 = (Cast)b2;
18         c3.m(); c3.n(); c3.o();
19     }
```

Explicit casting

Explicit casting

Explicit casting is based on a conception that programmer knows what he/she is doing

```
1  class A{}
2  class B extends A{};
3
4  public class UnsafeCast {
5      public static void main(String[] args){
6          A a = new A();
7          B b = (B)a;
8      }
9  }
```

Casting

```
1         Multiple m = new Multiple();
2         System.out.println(m instanceof Multiple);
3         System.out.println(Multiple.class.isInstance(m));
4         System.out.println(m instanceof Inherit);
5         System.out.println(m instanceof Base);
6         //System.out.println(m instanceof String);
7         System.out.println(String.class.isInstance(m));
8         System.out.println(m instanceof Object);
9
10        System.out.println();
11        Inherit i = new Inherit();
12        System.out.println(i instanceof Multiple);
13        System.out.println(Multiple.class.isInstance(i));
14        System.out.println(i instanceof Inherit);
15        System.out.println(i instanceof Base);
16        //System.out.println(i instanceof String);
17        System.out.println(String.class.isInstance(i));
18        System.out.println(i instanceof Object);
```

Casting

```
1         System.out.println();
2         Base b = new Base();
3         System.out.println(b instanceof Multiple);
4         System.out.println(Multiple.class.isInstance(b));
5         System.out.println(b instanceof Inherit);
6         System.out.println(b instanceof Base);
7         //System.out.println(b instanceof String);
8         System.out.println(String.class.isInstance(b));
9         System.out.println(b instanceof Object);
10
11        System.out.println();
12        b = m;
13        System.out.println(b instanceof Multiple);
14        System.out.println(Multiple.class.isInstance(b));
15        System.out.println(b instanceof Inherit);
16        System.out.println(b instanceof Base);
17        //System.out.println(b instanceof String);
18        System.out.println(String.class.isInstance(b));
19        System.out.println(b instanceof Object);
```

Safe way of casting

```
1  class A{}
2  class B extends A{};
3
4  public class UnsafeCast {
5      public static void main(String[] args){
6          A a = new A();
7          B b;
8
9          if (a instanceof B)
10             b = (B)a;
11     }
12 }
```

Field hiding

```
1  class First{
2      public int a = 3;
3  }
4
5  public class Hiding extends First {
6      public int a = 4;
7
8      public static void main(String[] args){
9          Hiding h = new Hiding();
10         System.out.println(h.a);
11         First f = new First();
12         System.out.println(f.a);
13
14         First g = h;
15         System.out.println(g.a);
16     }
```

4 3 3

Polymorphism

Polymorphism

It is a feature that one organism may have many different forms.

- Behaviour of object may vary in different cases
- Classes may have the same functionality but different behaviour
- Animal analogy: they are moving BUT some are running, others are swimming, others are flying

Polymorphism

```
1  class Bazowa1{
2      public void metoda(){
3          System.out.println("base method");
4      }
5  }
6  public class OverrideTest extends Bazowa1{
7      public void metoda(){
8          System.out.println("derived method");
9      }
10     public static void main(String[] args){
11         Bazowa1 b = new Bazowa1();
12         b.metoda();
13         OverrideTest o = new OverrideTest();
14         o.metoda();
15         b = o;
16         b.metoda();
17         o = (OverrideTest)b;
18         o.metoda();
19     }
```

Method overriding

- Method overriding
- Most important tool for polymorphism!
- Allow to dynamically determine which method should be invoked
- Late binding of methods

Example of polymorphism

```
1  class Animal{
2      public void makeSound(){}
3  }
4
5  class Dog extends Animal{
6      public void makeSound(){
7          System.out.println("Wuf Wuf");
8      }
9  }
10
11 class Cat extends Animal{
12     public void makeSound(){
13         System.out.println("Meow");
14     }
15 }
```

Example of polymorphism

```
1  class Cow extends Animal{
2      public void makeSound(){
3          System.out.println("Muuuu");
4      }
5  }
6
7  class MutatedCat extends Cat{
8      public void makeSound(){
9          System.out.println("Uuuuuu");
10     }
11 }
```

Example of polymorphism

```
1  public class OverrideExample {
2
3      public static void main(String[] args){
4          Animal[] animals = new Animal[5];
5          animals[0] = new Dog();
6          animals[1] = new MutatedCat();
7          animals[2] = new Cow();
8          animals[3] = new Cat();
9          animals[4] = new Cow();
10
11         for(Animal a: animals)
12             a.makeSound();
13     }
```

Risk

```
1  class BaseClass class{
2      public void method(){
3
4      }
5  }
6
7  public class OverrideAdnotation extends BaseClass {
8      public void mmethod(){
9          System.out.println("method");
10     }
11 }
```

Override annotation

```
1  class BazowaKlasa{
2      public void metoda(){
3
4      }
5  }
6
7  public class OverrideAdnotation extends BazowaKlasa {
8
9      @Override
10     public void metoda(){
11         System.out.println("metoda");
12     }
13
14     //@Override
15     public void mmetoda(){
16         System.out.println("metoda");
17     }
18 }
```

Overriding of static methods?

```
1  class BaseStaticOverride{
2      static void metoda1(){}
3      void metoda2(){}
4  }
5
6  public class StaticOverride extends BaseStaticOverride {
7      static void metoda1(){}
8  }
9
10 class StaticOverride1 extends BaseStaticOverride{
11     //void metoda1(){};
12 }
13
14 class StaticOverride2 extends BaseStaticOverride{
15     //static void metoda2(){}
16 }
17
18 class StaticOverride3 extends BaseStaticOverride{
19     void metoda2(){}
20 }
```

Overloading

- Allow to define multiple methods with the same name
- Simpler way to do polymorphism
- How to distinguish methods with the same names?

Overloading

```
1  public class Overload extends OverloadBase {
2
3      public void metoda(){
4          System.out.println("Without arguments");
5      }
6
7      public void metoda(int i){
8          System.out.println("int argument");
9      }
10
11     public void metoda(long l){
12         System.out.println("long argument");
13     }
14 }
```

```
1  class OverloadBase{
2      public void metoda(float f){
3          System.out.println("float argument");
4      }
5  }
6  ...
7      public static void main(String[] args){
8          Overload o = new Overload();
9          o.metoda();
10         o.metoda(1);
11         //o.metoda(12345678912345);
12         o.metoda(12345678912345L);
13         o.metoda(1.5f);
14     }
```

Overloading

```
1  public class Overload extends OverloadBase {
2
3      public void metoda(){
4          System.out.println("Bez argumentow");
5      }
6
7      //public int metoda(){}
8
9      public void metoda(int i){
10         System.out.println("Argument int");
11     }
12
13     public void metoda(long l){
14         System.out.println("Argument long");
15     }
16 }
```

Invoking methods from superclasess

```
1  class KlasaBazowa{
2      public void metoda(){
3          System.out.println("Metoda bazowa");
4      }
5  }
6
7  public class Super extends KlasaBazowa{
8
9      public void metoda(){
10         System.out.println("Metoda pochodna");
11     }
12
13     public static void main(String[] args){
14         Super s = new Super();
15         s.metoda();
16     }
17 }
```

Invoking methods from superclasses

```
1  class KlasaBazowa{
2      public void metoda(){
3          System.out.println("Meotda bazowa");
4      }
5  }
6
7  public class Super extends KlasaBazowa{
8
9      public void metoda(){
10         super.metoda();
11         System.out.println("Metoda pochodna");
12     }
13
14     public static void main(String[] args){
15         Super s = new Super();
16         s.metoda();
17     }
18 }
```

Invoking methods from superclasses

```
1  class KlasaBazowa{
2      public void metoda(){
3          System.out.println("Metoda bazowa");
4      }
5      public void metoda(int i){
6          System.out.println("Inna metoda bazowa");
7      }
8  }
9
10 public class Super extends KlasaBazowa{
11     public void metoda(){
12         super.metoda();
13         super.metoda(3);
14         System.out.println("Metoda pochodna");
15     }
16     public static void main(String[] args){
17         Super s = new Super();
18         s.metoda();
19     }
20 }
```

Accessing hidden fields

```
1  class KlasaBazowa{
2
3      int i = 5;
4
5  }
6
7  public class Super extends KlasaBazowa{
8
9      int i = 6;
10
11     public void metoda(){
12         System.out.println(i);
13         System.out.println(super.i);
14     }
15
16     public static void main(String[] args){
17         Super s = new Super();
18         s.metoda();
19     }
20 }
```

Constructors

Constructors

Constructors are not inherited but we can access constructors from superclasses.

```
1  class BaseConstructors{
2      BaseConstructors(){
3          System.out.println("Base constructor");
4      }
5  }
6
7  public class Constructors extends BaseConstructors {
8      Constructors(){
9          super();
10         //super();
11         System.out.println("Derived constructor");
12         //super();
13     }
14 }
```

Konstruktory

```
1  class BaseConstructors{
2      BaseConstructors(){
3          System.out.println("Base constructor");
4      }
5      BaseConstructors(int i){
6          System.out.println("Base constructor");
7      }
8
9  }
10
11 public class Constructors extends BaseConstructors {
12     Constructors(){
13         super();
14     }
15     Constructors(int i){
16         super(i);
17     }
18 }
```

final fields

```
1  class Bazowana{
2
3      public final int a = 4;
4      public final int b;
5
6      public Bazowana(){
7          b = 3;
8      }
9
10     //public Bazowana(float f){}
11
12     public void metoda(){
13         //a = 15;
14     }
15 }
```

final methods

```
1  class Bazowana{
2
3      public final void metoda(){
4      }
5
6  }
7  public class Final extends Bazowana {
8
9      //public void metoda(){}
10
11 }
```

final classes

```
1  final class Finalna{
2      public void metoda(){
3
4      }
5  }
6
7  //class Dziedziczaca extends Finalna{}
```

Inner classes

```
1  public class InnerClass {
2      class Bazowa{
3      }
4      class Pochodna extends Bazowa{
5      }
6      static class Inna{
7      }
8      public void test(){
9          Pochodna p = new Pochodna();
10     }
11     public static void main(String[] args){
12         //Pochodna p = new Pochodna();
13         InnerClass ic = new InnerClass();
14         Pochodna p = ic.new Pochodna();
15         Inna i = new InnerClass.Inna();
16     }
17 }
```

Inner classes

```
1  public class AnotherInnerClass extends InnerClass.Pochodna{
2      public AnotherInnerClass(InnerClass ic){
3          ic.super();
4      }
5
6      public static void main(String[] args){
7          InnerClass ic = new InnerClass();
8          AnotherInnerClass aic = new AnotherInnerClass(ic);
9      }
10 }
```

Types covariance

```
1  class Base1{
2      void method(){System.out.println("Bazowa");}
3  }
4  class Another extends Base1{
5      void method(){System.out.println("Pochodna");}
6  }
7  class BaseCovariance{
8      Base1 zwracanie(){return new Base1();}
9  }
10 public class Covariance extends BaseCovariance {
11     Another zwracanie(){return new Another();}
12     public static void main(String[] args){
13         BaseCovariance bc = new Covariance();
14         //Another a = bc.zwracanie();
15         Base1 b = bc.zwracanie();
16         b.method();
17     }
18 }
```

THE END!

Additional reading:

- https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html
- https://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html
- Thinking in Java chapters by Bruce Eckel:
  - Reusing Classes
  - Polymorphism