Object-oriented programming

Session 6

20th of March 2019
Ljubov Jaanuska
Classical example

```java
public class Dog {
    String name;
    Dog(String name) {
        this.name = name;
    }
    void voice(){
    }
}
```

http://en.wikipedia.org/wiki/Bark_(utterance)
```java
public class EstonianDog extends Dog {
    EstonianDog(String name) {
        super(name);
    }
    void voice(){
        System.out.println("Auh, auh!");
    }
}

public class LatvianDog extends Dog {
    LatvianDog(String name) {
        super(name);
    }
    void voice(){
        System.out.println("Vau, vau!");
    }
}

public class RussianDog extends Dog {
    RussianDog(String name) {
        super(name);
    }
    void voice(){
        System.out.println("Gav, gav!");
    }
}

public class FinnishDog extends Dog {
    FinnishDog(String name) {
        super(name);
    }
    void voice(){
        System.out.println("Vuh, vuh!"ера;
    }
}
```
public class TestDog {
    public static void main(String[] args){
        Dog k1 = new RussianDog("Sharik");
        Dog k2 = new LatvianDog("Reksis");
        Dog k3 = new FinnishDog("Musti");
        Dog k4 = new EstonianDog("Muri");
        Dog[] myDogs = {k1, k2, k3, k4};
        for (Dog k : myDogs){
            k.voice();
        }
    }
}
Is a method needed if it does nothing?

```java
public class Dog {
    String name;
    Dog(String name) {
        this.name = name;
    }
}
```
public class TestDog {
    public static void main(String[] args) {
        Dog k1 = new RussianDog("Sharik");
        Dog k2 = new LatvianDog("Reksis");
        Dog k3 = new FinnishDog("Musti");
        Dog k4 = new EstonianDog("Muri");
        Dog[] myDogs = {k1, k2, k3, k4};
        for (Dog k : myDogs) {
            k.voice();
        }
    }
}

java: cannot find symbol
   symbol: method voice()
   location: variable k of type Dog
public class TestDog {
    public static void main(String[] args) {
        Dog k1 = new RussianDog("Sharik");
        Dog k2 = new LatvianDog("Reksis");
        Dog k3 = new FinnishDog("Musti");
        Dog k4 = new EstonianDog("Muri");
        Dog[] myDogs = {k1, k2, k3, k4};
        for (Dog k : myDogs) {
            ((RussianDog) k). voice();
        }
    }
}

Gav, gav!
Exception in thread "main"
java.lang.ClassCastException: class LatvianDog
cannot be cast to class RussianDog
at TestDog.main(TestDog.java:10)
public class TestDog {
    public static void main(String[] args){
        Dog k1 = new RussianDog("Sharik");
        Dog k2 = new LatvianDog("Reksis");
        Dog k3 = new FinnishDog("Musti");
        Dog k4 = new EstonianDog("Muri");
        Dog[] myDogs = {k1, k2, k3, k4};
        for (Dog k : myDogs){
            if (k instanceof RussianDog)
                ((RussianDog)k).voice();
            if (k instanceof LatvianDog)
                ((LatvianDog)k).voice();
        }
    }
}
Method that does nothing

```java
public class Dog {
    String name;
    Dog(String name) {
        this.name = name;
    }
    void voice() {
    }
}
```
So...

- A subclass is derived from a superclass adding
  - new instance fields
  - new methods
  - overriding
- To avoid doubling, define object features at the highest level
- If a superclass is so abstract that methods do not have bodies define
  - abstract method, abstract class
- Java allows only one superclass
  - and several interfaces
Abstract methods

• a method without body
• only semicolon (;)
• modifier abstract
• abstract method cannot at the same time private, final nor static

abstract data_type name(form. params);

abstract void voice();
Abstract class

• a class which contains abstract methods
• modifier `abstract`
• is used as a superclass
• a subclass should implement all abstract methods of its superclass; otherwise, the subclass should also be defined as an abstract class
abstract public class Dog {
    String name;
    Dog(String name) {
        this.name = name;
    }
    abstract void voice();
}
Abstract class

• A subclass can be abstract even if its superclass is not abstract (e.g. class `Object` is not abstract)

• It is not possible to create objects of abstract class; however, instances may have a data type of an abstract class

```java
class DogTest {
    public static void main(String[] args) {
        Dog k5 = new EstonianDog("Muri");
        //Dog k7 = new Dog("Lassie"); //ERROR
    }
}
```
Why should a class be abstract?

• A subclass is forced to inherit features of its abstract superclass and implement superclass’s abstract methods
To avoid doubling, implement object features at the highest possible level.
Module java.base
Package java.util

Class Calendar

java.lang.Object
  java.util.Calendar

All Implemented Interfaces:
Serializable, Cloneable, Comparable<Calendar>

Direct Known Subclasses:
GregorianCalendar

public abstract class Calendar
extends Object
implements Serializable, Cloneable, Comparable<Calendar>
Should an abstract class and an abstract method have a modifier?

1. No
2. Yes
3. Only abstract class should have
4. Only abstract method should have
Can an abstract class have non-abstract methods?

1. yes
2. no
Can an abstract class have non-abstract methods? Can non-abstract methods be invoked?

```java
public abstract class K {
    abstract void method1();
    void method2() {
        System.out.println("Class K");
    }
}

public class KK extends K {
    void method1() {
        System.out.println("Class KK");
        method2();
    }
}

public class TestKK {
    public static void main(String[] args) {
        K k = new KK();
        k.method1();
        k.method2();
    }
}
```
Can an abstract method be private?

1. yes
2. no
public abstract class Mammals {
    private String color;
    private int lifeSpan;

    public String getColor() { return color; }
    public void setColor(String color) {
        this.color = color;
    }

    public int getLifeSpan() { return lifeSpan; }
    public void setLifeSpan(int lifeSpan) {
        this.lifeSpan = (test(lifeSpan)) ? lifeSpan : 0;
    }

    private boolean test(int num) { return num >= 0; }

    public abstract void reproduction();
}
A subclass should implement abstract methods of its superclass

```java
abstract public class Dog {
    String name;
    Dog (String name) {
        this.name = name;
    }
    abstract void voice();
}

public class EstonianDog extends Dog {
    EstonianDog(String name) {
        super(name);
    }
    void voice() {
        System.out.println("Auh, auh!");
    }
}
```
A subclass should implement abstract methods of its superclass or a subclass should also be abstract

```java
abstract public class Dog {
    String name;
    Dog (String name) {
        this.name = name;
    }
    abstract void voice();
}

abstract public class BigDog extends Dog {
    BigDog(String name) {
        super(name);
    }
}
```
Interfaces

• An interface is a set of abstract methods; these methods are implemented in classes which promise to implement the whole interface
  – An interface may have constants, default and private methods

 modifier interface InterfaceName { ... }

• To implement an interface, a class should have keyword implements
  – which shows that a class promises to implement all abstract methods
Interface

• To implement an interface, a class should implement all abstract methods; otherwise, the class should be abstract

• An interface has
  – methods **public abstract**
  – constants **public static final**

however, modifiers are not compulsory
public interface Eatable {
    public String howEat();
}

public class Rice implements Eatable {
    public String howEat() {
        return "boiled";
    }
}

public class Potato implements Eatable {
    public String howEat() {
        return "boiled or fried";
    }
}
What is the output?

```java
public class TestEatable{
    public static void main(String[] args) {
        System.out.println(new Rice().howEat());
    }
}
```

1. boiled
2. boiled or fried
3. nothing
4. something else
5. error
What is the output?

```java
public class TestEatable{
    public static void main(String[] args) {
        Eatable t1 = new Rice();
        System.out.println(t1.howEat());
    }
}
```

1. boiled
2. boiled or fried
3. nothing
4. something else
5. error
What is the output?

```java
public class TestEatable{
    public static void main(String[] args) {
        Eatable t1 = new Eatable();
        System.out.println(t1.howEat());
    }
}
```

1. boiled
2. boiled or fried
3. nothing
4. something else
5. error
public interface Xline {
    int num = 10;  //public static final
    //private String s = "Error!";

    void method1();  //public abstract

    default void method3() {
        System.out.println("Default task");
    }

    private void method4() {
        System.out.println("Help");
    }

    /*
    *void method4() {
        System.out.println("Error!");
    }
    */
}
Interface Comparable\(<T>\>

Type Parameters:

T - the type of objects that this object may be compared to

All Known SubInterfaces:

AnnotationTypeDoc, AnnotationTypeElementDoc, ArrayType, ByteValue, CharValue, 
ExecutableMemberDoc, Field, FieldDoc, FloatValue, IntegerValue, InterfaceType, 
RunnableScheduledFuture\(<V>\), ScheduledFuture\(<V>\), SerialFieldTag, ShortValue

All Known Implementing Classes:

AbstractChronology, AbstractRegionPainter.PaintContext.CacheMode, AccessMode, 
BigDecimal, BigInteger, Boolean, Byte, ByteBuffer, Calendar, CardTerminals.State, 
ClientInfoStatus, CollationKey, Collector.Characteristics, Component.Baseline, 
Diagnostic.Kind, Diagnostic.Kind, Dialog.ModalExclusionType, Dialog.ModalityType, 
Duration, ElementKind, Elements.Origin, ElementType, Enum, File, FileTime, FileVirtualFile, 
GraphicsDevice.WindowTranslucency, GregorianCalendar, GroupLayout.Alignment, 
IsoEra, JapaneseChronology, JapaneseDate, JavaFileObject.Kind, JConsoleContext, 
LinkOption, LocalDate, LocalDateTime, Locale.Category, Locale.FilteringMode, 
Locale.PlainText
public class TwoNum{
    private int a;
    private int b;

    public TwoNum(int a, int b) {
        this.a = a;
        this.b = b;
    }

    int sum(){
        return a + b;
    }

    public String toString() {
        return "TwoNum[a=" + a + ", b=" + b + ", sum()=" + sum() + "]";
    }
}
public class TestTwoNum{
    public static void main(String[] args) {
        TwoNum ka1 = new TwoNum(4, 3);
        TwoNum ka2 = new TwoNum(5, 0);
        TwoNum ka3 = new TwoNum(2, 4);
        TwoNum [] kad = {ka1, ka2, ka3};
        for (TwoNum ka : kad)
            System.out.println(ka);
    }
}
public class TwoNum
    implements Comparable<TwoNum>{

@Override
public int compareTo(TwoNum o) {
    if (this.sum() > o.sum()) {
        return 1;
    }
    if (this.sum() < o.sum()) {
        return -1;
    }
    return 0;
}
java.util.Arrays.sort(kad);
for (TowNum ka : kad)
    System.out.println(ka);

TwoNum[a=5, b=0, sum()=5]
TwoNum[a=2, b=4, sum()=6]
TwoNum[a=4, b=3, sum()=7]
public int compareTo(TwoNum o) {
    return this.sum() - o.sum();
}

public int compareTo(TwoNum o) {
    return Integer.compareTo(this.sum(), o.sum());
}

public int compareTo(TwoNum o) {
    return Integer.valueOf(this.sum()).compareTo(
        Integer.valueOf(o.sum()));
}
public class Person {
    private String name;
    private int length;
    public Person(String name, int length) {
        this.name = name;
        this.length = length;
    }
    public String toString() {
        return "Person[name=\" + name + ", length=\" + length + "]";
    }
}
public class TestPerson {
    public static void main(String[] args) {
        Person p1 = new Person("Eva", 160);
        Person p2 = new Person("Jane", 170);
        Person p3 = new Person("Ada", 165);
        Person[] id = {p1, p2, p3};
        for (Person i: id)
            System.out.println(i);
    }
}
public class Person implements Comparable<Person> {

    public int compareTo(Person p2) {

        //String
        if (name.compareTo(p2.name) != 0)
            return perenimi.compareTo(p2.name);

        //int
        return Integer.compare(length, p2.length);
    }
}
List

Module java.base
Package java.util

Interface List<E>

Type Parameters:
E - the type of elements in this list

All SuperInterfaces:
Collection<E>, Iterable<E>

All Known Implementing Classes:
AbstractList, AbstractSequentialList, ArrayList, AttributeList,

public interface List<E>
extends Collection<E>

An ordered collection (also known as a sequence). The user of this inte
Still interfaces

• A class can implement several interfaces; interfaces may have methods with the same names

    implements Interface1, Interface2

• An interface may have subinterfaces (own hierarchy)
Does it work?

```java
public interface Test{
    String t = "test test test";
    public void think(){
        System.out.println(t);
    }
}
```

1. yes
2. no
## Abstract class

- Class
- Essence
- extends
- Abstract and non-abstract methods
- Modifier `abstract` is compulsory
- Constructors, instance fields
- Inheritance – one class

## Interface

- Data type
- Property
- `implements`
- Abstract, default and private methods
- Modifier `abstract` is not compulsory
- Cannot have constructors nor instance fields
- One class can implement several interfaces
What is the output?

```java
public class A {
    public int method1() {
        return 1;
    }
}

public interface B {
    default int method1() {
        return 5;
    }
}

public class C extends A implements B {
}

C c1 = new C();
System.out.println(c1.method1());
```

1. 1
2. 5
3. nothing
4. error
public class A {
    public int method1() {
        return 1;
    }
}

public interface B {
    default int method1() {
        return 5;
    }
}

public class C extends A implements B {
}

B c1 = new C();
System.out.println(c1.method1());

1. 1
2. 5
3. nothing
4. error
What is the output?

```java
public interface A {
    public int method1(){
        return 1;
    }
}
public interface B {
    default int method1(){
        return 5;
    }
}
public class C implements A, B {
}
C c1 = new C();
System.out.println(c1.method1());
```

1. 1
2. 5
3. nothing
4. error
Inheritance

- One superclass
- „Diamond problem“

https://en.wikipedia.org/wiki/Multiple_inheritance
https://courses.cs.ut.ee/2019/oopn/spring/Main/S6