CS 102

Object Oriented Programming

Inheritance and Polymorphism

Reyyan Yeniterzi

reyyan.yeniterzi@ozyegin.edu.tr

Announcements

- Assignment 1 grades are available @LMS.
- Check your grade and comment
  - Any questions and objections, contact to your TA Osman
- Assignment 2 is due tonight!!!
  - Do not make the same mistakes you did in Assignment 1.
  - Be careful about submission instructions.
  - Check them and check them again.

- Let's see more benefits of inheritance

- Assume that we have an animal farm with different types of animals and we don't know inheritance

Cat

```java
public class Cat {
    private String name;
    private String color;

    public Cat (String name) {
        this.name = name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
    }
    public String getColor() {
        return color;
    }
    public String speak() {
        return "Miyauv";
    }
}
```

Dog

```java
public class Dog {
    private String name;
    private String color;

    public Dog (String name) {
        this.name = name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
    }
    public String getColor() {
        return color;
    }
    public String speak() {
        return "Woof";
    }
}
```

- Assume that we have an animal farm with different types of animals and we don't know inheritance.
- Can we store all these different animal types in one data structure, like an array?

- Dog
- Cat
- Cow
- Mouse

Serafettin
Scooby
Sarı Kız
Rin Tin Tin
Tom
Jerry

- Assume that we have an animal farm with different types of animals and we don't know inheritance.
- Can we store all these different animal types in one data structure, like an array?

- Dog
- Cat
- Cow
- Mouse

Serafettin
Scooby
Sarı Kız
Rin Tin Tin
Tom
Jerry

- An array needs to hold objects of same type!

- Assume that we have an animal farm with different types of animals and we don't know inheritance.
- Can we store all these different animal types in one data structure, like an array?

- Dog
- Cat
- Cow
- Mouse

Serafettin
Scooby
Sarı Kız
Rin Tin Tin
Tom
Jerry

- An array needs to hold objects of same type!

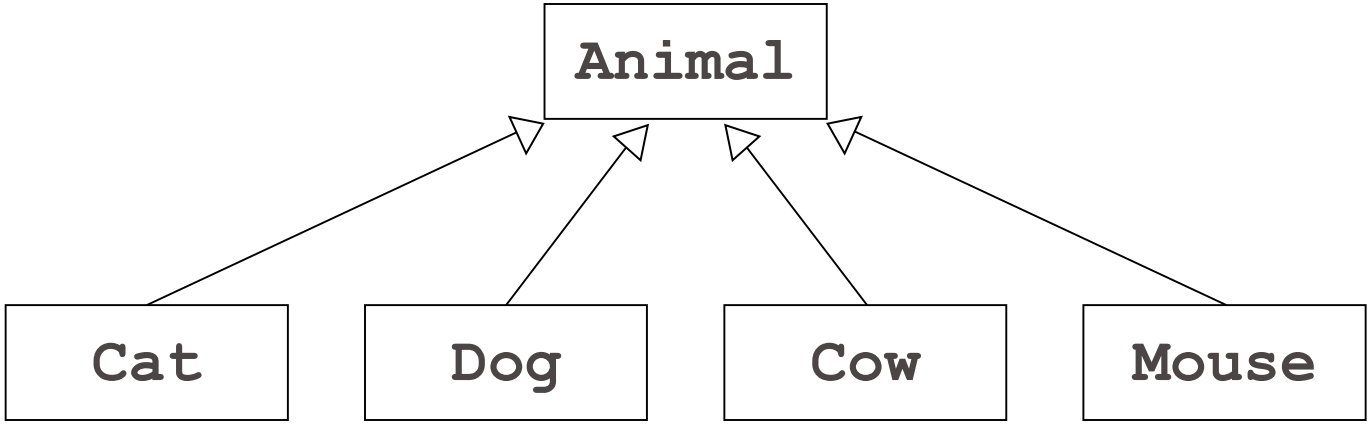
Serafettin
Tom

Jerry

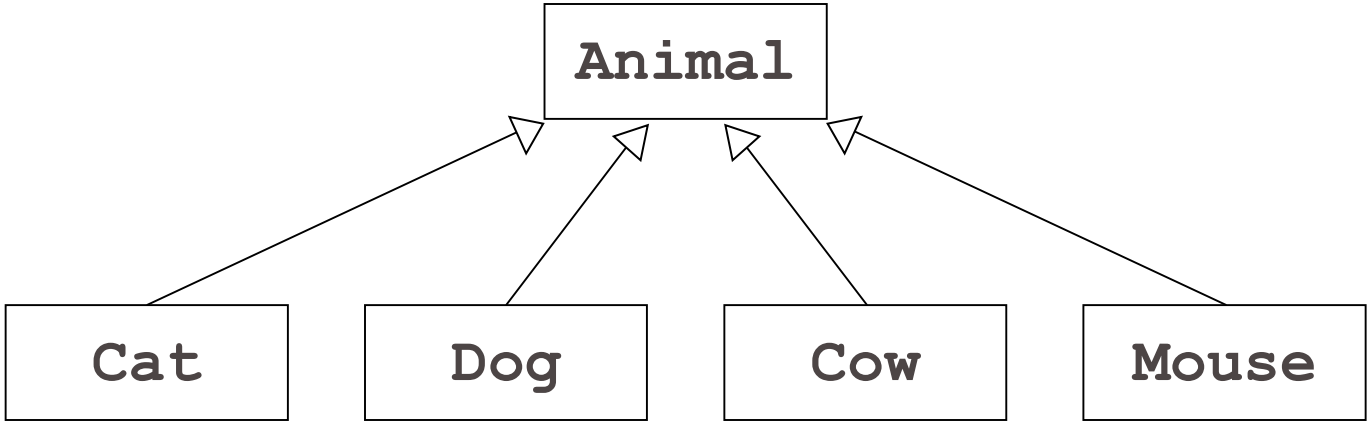
Sarı Kız

Scooby
Rin Tin Tin

- Inheritance gives us the ability to store all animals in one data structure.



- Inheritance gives us the ability to store all animals in one data structure.



- A cat/dog/cow/mouse is an Animal

Reference and Object


```
Animal animal1 = new Animal();
```

Reference and Object


```
Animal animal1 = new Animal();
```

Reference

Object

Reference and Object

Reference Object

```
Animal animal1 = new Animal();
```

Animal **is an** animal ✓

Reference and Object

Reference Object

```
Cat cat1 = new Cat("Serafettin");

Dog dog1 = new Dog("Scooby");
```

Are these statements legal?

Reference and Object

Reference Object

```
Cat cat1 = new Cat("Serafettin");

Dog dog1 = new Dog("Scooby");
```

Cat **is a** cat

Dog **is a** dog

Both of them are legal statements.

Reference and Object

Reference Object

```
Animal animal2 = new Dog("Rin Tin Tin");

Animal animal3 = new Cat("Tom");
```

Are these statements legal?

Can an animal reference point to a cat/dog object?

Reference and Object

Reference Object

```
Animal animal2 = new Dog("Rin Tin Tin");

Animal animal3 = new Cat("Tom");
```

Cat **is an** animal

Dog **is an** animal

cat/dog has all the capabilities of an animal since it is derived from the animal

Both of them are legal statements.

Inheritance

- Therefore we can keep all animal types in one single data structure.
- An animal array can store an animal object and other animal types (cat, dog etc.).

```
Animal[] animals = new Animal[3];
animals[0] = new Animal();
animals[1] = new Cat("Tom");
animals[2] = new Dog("Rin Tin Tin");
```

Reference and Object

Reference Object

```
Cat cat2 = new Animal();
```

Is this a legal statement?

Reference and Object

Reference Object

```
Cat cat2 = new Animal();
```

Not a legal statement

Not all animals is a cat.

An animal may not have all the capabilities of a cat.

Cats can jump but not all animals can.

Reference and Object

Reference Object

```
Object object1 = new Cat("Tom");
```

Is this a legal statement?

Reference and Object

Reference Object

```
Object object1 = new Cat("Tom");
```

Cat **is an** object.

This is a legal statement.

```java
public class Animal {

    private String name;
    private String color;

    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
    }
    public String getColor() {
        return color;
    }
}
```

```java
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
    public String speak() {
        return "Miyauv";
    }
}
```

```java
Animal animal1 = new Animal();
Cat cat1 = new Cat("Serafettin");

String color = cat1.getColor();
animal1 = cat1;
```

- Are these statements valid?

```java
public class Animal {

    private String name;
    private String color;

    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
    }
    public String getColor() {
        return color;
    }
}
```

```java
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
    public String speak() {
        return "Miyauv";
    }
}
```

```java
Animal animal1 = new Animal();
Cat cat1 = new Cat("Serafettin");

String color = cat1.getColor();
animal1 = cat1;
```

- An animal reference can refer to a cat object.
- All cats are animal.

```java
public class Animal {

    private String name;
    private String color;

    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
    }
    public String getColor() {
        return color;
    }
}
```

```java
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
    public String speak() {
        return "Miyauv";
    }
}
```

```java
Animal animal1 = new Animal();
Cat cat1 = new Cat("Serafettin");

String color = cat1.getColor();
animal1 = cat1;
animal1.speak();
```

```java
public class Animal {

    private String name;
    private String color;

    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
    }
    public String getColor() {
        return color;
    }
}
```

```java
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
    public String speak() {
        return "Miyauv";
    }
}
```

```java
Animal animal1 = new Animal();
Cat cat1 = new Cat("Serafettin");

String color = cat1.getColor();
animal1 = cat1;
animal1.speak();
```

The method speak() is undefined for the type Animal

2 quick fixes available:

- Create method 'speak()' in type 'Animal'
- Add cast to 'animal1'

Press 'F2' for focus

- animal1 is an animal reference
- Compiler knows the reference type but not the object type

```java
public class Animal {

    private String name;
    private String color;

    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String toString() {
        return "Animal";
    }
}
```

```java
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
    public String toString() {
        return "Cat";
    }
}
```

```java
public class Dog extends Animal {
    public Dog(String name) {
        setName(name);
        setColor("black");
    }
    public String toString() {
        return "Dog";
    }
}
```

- What is the output?

```java
Animal[] animals = new Animal[3];
animals[0] = new Animal();
animals[1] = new Cat("Tom");
animals[2] = new Dog("Rin Tin Tin");

for (int i = 0; i < animals.length; i++) {
    System.out.println(animals[i]);
}
```

```java
public class Animal {

    private String name;
    private String color;

    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String toString() {
        return "Animal";
    }
}
```

```java
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
    public String toString() {
        return "Cat";
    }
}
```

```java
public class Dog extends Animal {
    public Dog(String name) {
        setName(name);
        setColor("black");
    }
    public String toString() {
        return "Dog";
    }
}
```

```java
Animal[] animals = new Animal[3];
animals[0] = new Animal();
animals[1] = new Cat("Tom");
animals[2] = new Dog("Rin Tin Tin");

for (int i = 0; i < animals.length; i++) {
    System.out.println(animals[i]);
}
```

```
Animal
Cat
Dog
```

Polymorphism

- Polymorphism gives us the capability to call the right method.

Compile Time vs. Runtime

- What does compiler do?
  - Compiler interprets our code
- Then what happens in runtime?
  - At runtime, the environment executes the interpreted code

Compile Time vs. Runtime

- What does compiler do?
  - Compiler interprets our code
- Then what happens in runtime?
  - At runtime, the environment executes the interpreted code

- There are two steps of our programs:
  - Compiler time
  - Runtime

Compile Time vs. Runtime

- Compile time decisions are based on reference type
- Run time decisions are based on object type

Compiler

- Only knows about the reference type.
- When a method is called, it looks for that method inside that particular reference type class.
- Example in the next slide...

```java
public class Animal {

    private String name;
    private String color;

    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
    }
    public String getColor() {
        return color;
    }
}
```

```java
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
    public String speak() {
        return "Miyauv";
    }
}
```

```java
Animal animal1 = new Animal();
Cat cat1 = new Cat("Serafettin");

String color = cat1.getColor();
animal1 = cat1;
animal1.speak();
```

The method speak() is undefined for the type Animal

2 quick fixes available:

- Create method 'speak()' in type 'Animal'
- Add cast to 'animal1'

Press 'F2' for focus

Runtime

- At runtime, the exact runtime object is used to find where a method belogs to.
- The method used needs to match with the signature of the actual method.
- Example in the next slide...

```java
public class Animal {

    private String name;
    private String color;

    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String toString() {
        return "Animal";
    }
}
```

```java
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
    public String toString() {
        return "Cat";
    }
}
```

```java
public class Dog extends Animal {
    public Dog(String name) {
        setName(name);
        setColor("black");
    }
    public String toString() {
        return "Dog";
    }
}
```

- toString() method
- The method signatures match

```java
Animal[] animals = new Animal[3];
animals[0] = new Animal();
animals[1] = new Cat("Tom");
animals[2] = new Dog("Rin Tin Tin");

for (int i = 0; i < animals.length; i++) {
    System.out.println(animals[i]);
}
```

```
Animal
Cat
Dog
```

```java
public class Animal {

    private String name;
    private String color;

    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
    }
    public String getColor() {
        return color;
    }
}
```

```java
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
    public String speak() {
        return "Miyauv";
    }
}
```

```java
Animal animal1 = new Animal();
Cat cat1 = new Cat("Serafettin");

String color = cat1.getColor();
animal1 = cat1;
animal1.speak();
```

The method speak() is undefined for the type Animal

2 quick fixes available:

- Create method 'speak()' in type 'Animal'
- Add cast to 'animal1'

Press 'F2' for focus

- Is there a way to fix this?
- Lets assume that animal1 will always refer to a cat object?

```java
public class Animal {

    private String name;
    private String color;

    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
    }
    public String getColor() {
        return color;
    }
}
```

```java
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
    public String speak() {
        return "Miyauv";
    }
}
```

```java
Animal animal1 = new Animal();
Cat cat1 = new Cat("Serafettin");

String color = cat1.getColor();
animal1 = cat1;
animal1.speak();
```

The method speak() is undefined for the type Animal

2 quick fixes available:

- Create method 'speak()' in type 'Animal'
- Add cast to 'animal1'

Press 'F2' for focus

- It is possible with explicit casting

Casting

- Widening
  - Automatic type promotion (from int to double)

```
int a = 5;
double b = a;
System.out.println(b);
```

Casting

- Widening
  - Automatic type promotion (from int to double)

```
int a = 5;
double b = a;
System.out.println(b);
```

  - Superclass reference = subclass object;

```
Animal animal2 = new Cat("Tom");
```

Casting

- Narrowing
  - Explicit casting (from double to int)

```
double c = 9.99;
int d = ((int) c);
System.out.println(d);
```

Casting

- Narrowing
  - Explicit casting (from double to int)

```
double c = 9.99;
int d = ((int) c);
System.out.println(d);
```

  - Subclass reference = (subclass) superclass reference;

```
((Cat) animal1).speak();
```

  - Be careful, compiler trusts you at this point

```java
public class Animal {

    private String name;
    private String color;

    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
    }
    public String getColor() {
        return color;
    }
}
```

```java
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
    public String speak() {
        return "Miyauv";
    }
}
```

- Compiler will search for speak method inside the cat class.

```java
Animal animal1 = new Animal();
Cat cat1 = new Cat("Serafettin");

animal1 = cat1;
//animal1.speak(); // does not work
((Cat) animal1).speak();
```

```java
public class Animal {

    private String name;
    private String color;

    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
    }
    public String getColor() {
        return color;
    }
}
```

```java
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
    public String speak() {
        return "Miyauv";
    }
}
```

- What do you think will happen at this point?

```java
Animal animal1 = new Animal();
Cat cat1 = new Cat("Serafettin");

//animal1 = cat1;
//animal1.speak(); // does not work
((Cat) animal1).speak();
```

Casting

- We won't get a compiler error, relax

```
Animal animal1 = new Animal();
Cat cat1 = new Cat("Serafettin");

//animal1 = cat1;
//animal1.speak(); // does not work
((Cat) animal1).speak();
```

Casting

- We won't get a compiler error, relax

```
Animal animal1 = new Animal();
Cat cat1 = new Cat("Serafettin");

//animal1 = cat1;
//animal1.speak(); // does not work
((Cat) animal1).speak();
```

- It will be worse, we will get a runtime error

```
Exception in thread "main" java.lang.ClassCastException: AnimalFarmV05.Animal cannot be cast to AnimalFarmV05.Cat
        at AnimalFarmV05.AnimalFarm.main(AnimalFarm.java:51)
```

Casting

- We need to make sure that we don't cast wrong.
- How?

Casting

- We need to make sure that we don't cast wrong.
- How?
  - By doing runtime type check
  - instanceof operator
    - Checks whether there is an is a relationship

```
Animal animal1 = new Animal();
Cat cat1 = new Cat("Serafettin");

//animal1 = cat1;
//animal1.speak(); // does not work
if (animal1 instanceof Cat) {
    ((Cat) animal1).speak();
}
```

Final method

- A method in the superclass that cannot be overridden in a subclass.
- Any idea which methods can be final?

Final method

- A method in the superclass that cannot be overridden in a subclass.
- Methods that are declared private are implicitly final, because it's not possible to override them in a subclass.
- Methods that are declared static are implicitly final.

Static vs. Dynamic Binding

- A final method's declaration can never change, so all subclasses use the same method implementation, and calls to final methods are resolved at compile time—this is known as **static (early) binding**.
- **Dynamic (late) binding**: methods to be executed are determined in runtime, depending on the object type.

What is the output?


```java
public class Animal {

    private String name;
    private String color;

    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
        }
    public String getColor() {
        return color;
    }
    public String speak() {
        return "Some Noise";
    }
    public String toString() {
        return "Animal " + this.getName() +
                " is in color " + this.getColor() +
                " and speaks " + this.speak();
    }
}
```

```java
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
    public String speak() {
        return "Miyauv";
    }
}
```

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

    Animal animal = new Cat("Tom");
    System.out.println(animal);
}
```

What is the output?


```java
public class Animal {

    private String name;
    private String color;

    public void setName(String name) {
```

```
Animal Tom is in color gray and speaks Miyauv
```

```java
    }
    public String getName() {
        return name;
        }
    public String getColor() {
        return color;
    }
    public String speak() {
        return "Some Noise";
    }
    public String toString() {
        return "Animal " + this.getName() +
                " is in color " + this.getColor() +
                " and speaks " + this.speak();
    }
}
```

```java
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
    public String speak() {
        return "Miyauv";
    }
}
```

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

    Animal animal = new Cat("Tom");
    System.out.println(animal);
}
```

What is the output?


```java
public class Animal {

    private String name;
    private String color;

    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
        }
    public String getColor() {
        return color;
    }
    public String speak() {
        return "Some Noise";
    }
    public String toString() {
        return "Animal " + this.getName() +
                " is in color " + this.getColor() +
                " and speaks " + this.speak();
    }
}
```

What is the output?


```java
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
    public String speak() {
        return "Miyauv";
    }
    public String toString() {
        return "Cat " + this.getName() +
                " is in color " + this.getColor() +
                " and speaks " + super.speak();
    }
}
```

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

    Animal animal = new vanCat("Tom");
    System.out.println(animal);
}
```

```java
public class vanCat extends Cat{

    public vanCat(String name) {
        super(name);
        setColor("white");
    }
}
```

What is the output?


```java
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
    public String speak() {
        return "Miyauv";
    }
    public String toString() {
        return "Cat " + this.getName() +
                " is in color " + this.getColor() +
                " and speaks " + super.speak();
    }
}
```

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

    Animal animal = new vanCat("Tom");
    System.out.println(animal);
}
```

```java
public class vanCat extends Cat{

    public vanCat(String name) {
        super(name);
        setColor("white");
    }
}
```

What is the output?


```java
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
    public String speak() {
        return "Miyauv";
    }
    public String toString() {
        return "Cat " + this.getName() +
                " is in color " + this.getColor() +
                " and speaks " + super.speak();
    }
}
```

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

    Animal animal = new vanCat("Tom");
    System.out.println(animal);
}
```

```java
public class vanCat extends Cat{

}
```

```
Cat Tom is in color white and speaks Some Noise
```

- Call to the super.someMethod() get bound at compile time.
- Call to the this.someMethod() get bound at runtime.

Any Questions ?