Java Programming 2
Inheritance

Mary Ellen Foster
MaryEllen.Foster@glasgow.ac.uk
Semester 1 2020/2021

Inheritance

Objects (world or software) have some features in common

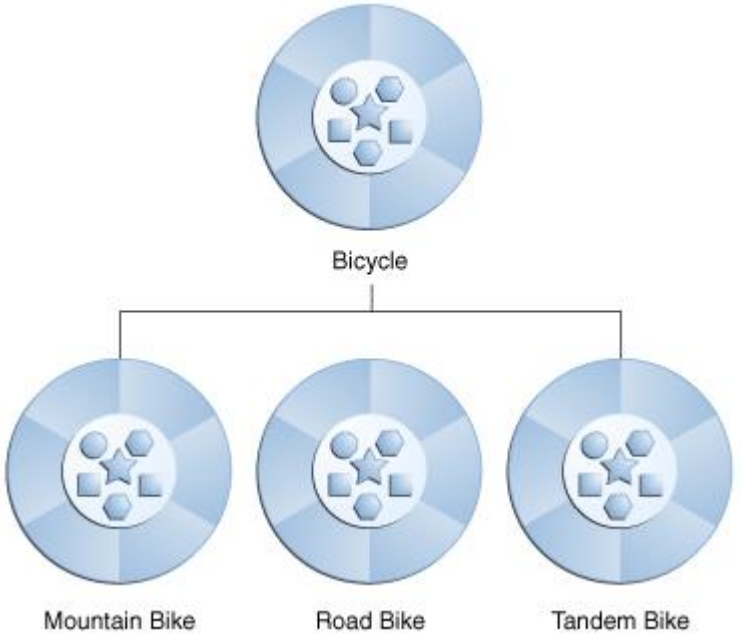
In OO programming, classes can **inherit** state and behaviour (fields and methods) from other classes

- Subclass is a **specialised version** of the superclass

In Java, a class can have **exactly one** superclass

- If superclass isn't specified, then it inherits from `Object`

Subclasses can **override** superclass methods to provide specialised behaviour



Java Tutorial "What is Inheritance?"

Inheritance in Java

```
public class Dog extends Animal {
  …
}
```

Java supports **single inheritance** (a class can extend at most one class)

One of the most powerful object-oriented features of Java

Root of Java class hierarchy is `java.lang.Object`

Full example

```
public class Animal {
    protected String name;

    public void move() {
        System.out.println(name
              +  “can move”);
    }
    public String getName() {
        return this.name;
    }
}
```

```
public class Dog extends Animal {
    private String breed;

    public void move() {
        System.out.println(name
            + “can walk and run”);
    }
}
```

What you can do in a subclass (Fields)

Use the inherited fields just like other fields *(except if they are private)*

Declare a field in the subclass with the same name as in the superclass

This is known as **hiding** the parent field (not recommended!)

Declare new fields that are not in the superclass

What you can do in a subclass (Methods)

Use the inherited methods directly *(unless they are private)*

Override an instance method by writing a new method with the same signature

Hide a static method by writing a new method with the same signature

Declare new methods that are not in the superclass

Method overriding

If a subclass has an instance method with an **identical** signature (name, parameters, return type*) as a parent class …

… then the subclass method **overrides** the parent method

This is a method for specifying alternative (specialised) behaviour for the subclass

```
public class Animal {
    public void move() {
        System.out.println("Animals can move");
    }
}

public class Dog extends Animal {
    public void move() {
        System.out.println("Dogs can walk and run");
    }
}
```

https://www.tutorialspoint.com/java/java_overriding.htm

* Actually, the subclass method's return type could also be a subclass of the parent class method's return type – this is known as a **covariant return type**.

Polymorphism

Literal meaning: “many forms”

In OO design: whenever an instance of class A is expected, you can also use an instance of any subclass of A

If a method is overridden in a subclass, Java will always use the most-specific overridden version

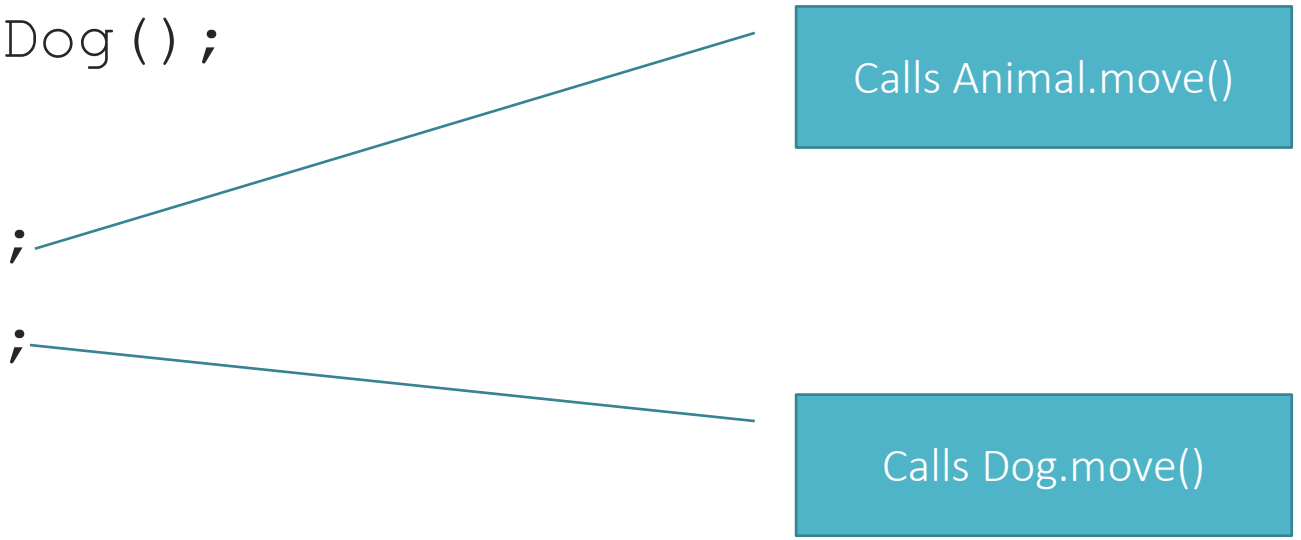
Even when the variable type is the superclass

Supported by **virtual method invocation**: method calls are dynamically dispatched based on the **runtime** type of the receiver object

Polymorphism example

```
Animal a1, a2;
a1 = new Animal();
a2 = new Dog();

a1.move();
a2.move();
```



Accessing superclass methods

But what if you really want to use the non-overridden method sometimes?

Use the `super` keyword (similar to `this`, but refers to super-class implementation)

```
public class Superclass {
    public void printMethod() {
        System.out.println("Printed in Superclass.");
    }
}
```

```
public class Subclass extends Superclass {
    // overrides printMethod in Superclass
    public void printMethod() {
        super.printMethod();
        System.out.println("Printed in Subclass");
    }
    public static void main(String[] args) {
        Subclass s = new Subclass();
        s.printMethod();
    }
}
```

Built-in methods

All objects inherit some methods from the root class `java.lang.Object`

- `toString()`: generates a `String` representation of the object
- `equals()`: compares two objects for equality and returns a `boolean` result
- `hashCode()`: returns an integer associated with the class for use in more complex data structures

You can override these methods to provide class-specific behaviour

toString() for BankAccount

Default behaviour: prints the class and the memory address of the BankAccount (rarely useful)

Overridden behaviour: prints the information in the relevant bank account fields

```
public String toString() {
    return "Account #" + id
           + " Name=" + name
           + ", balance="
           + balance;
}
```