Additional Notes on: Interfaces & Polymorphism

Further required readings from Liang (Textbook): Chapter 11 (Polymorphism)
Chapter 13 (Interfaces)

Credits: **http://www.ldodds.com/**

Overview

- Interfaces
- Static Binding
- Dynamic Binding
  - Polymorphism
- Casting

Interfaces

- Interfaces define a contract
  - Contain methods and their signatures
  - Can also include static final constants (and comments)
  - But no implementation
- Very similar to abstract classes
- One interface can extend another, but
  - An interface *cannot* extend a class
  - A class *cannot* extend an interface
  - Classes *implement* an interface

Defining Interfaces

- Use the `interface` keyword

```
public interface Vehicle {
public void turnLeft();
public void turnRight();
}
```

- Like abstract methods, the signature is terminated with a semi-colon

Implementing Interfaces

- Use the `implements` keyword

```
public class MotorBike implements Vehicle {
//include methods from Vehicle interface
}
```

- Class must implement all methods of the interface
  - *OR* declare itself to be `abstract`
- Classes can implement any number of interfaces

```
public class MotorBike implements Vehicle, Motorised
```

- Possible to combine extends and implements

```
public class MotorBike extends WheeledVehicle
   implements Vehicle, Motorised
```

Benefits of Interfaces

- Cleanly separates implementation of behaviour from description of the behaviour
  - Means the implementation is easily changed

```
Vehicle vehicle = new MotorBike();
// might become…
Vehicle vehicle = new MotorCar();
```

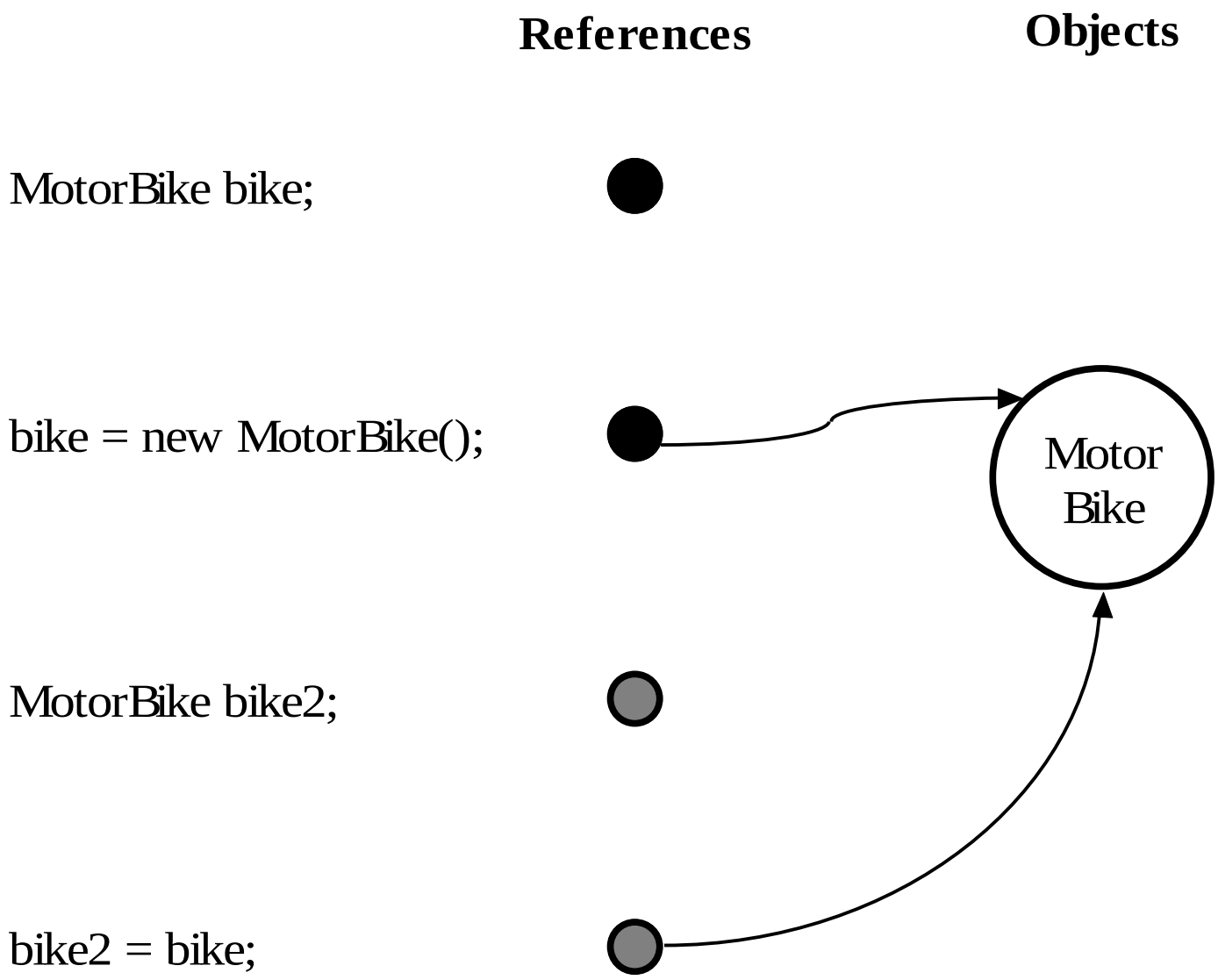
- Many OO systems are defined almost entirely of interfaces
  - Describes how the system will function
  - The actual implementation is introduced later

Overview

- Interfaces
- Static Binding
- Dynamic Binding
  - Polymorphism
- Casting

Binding

- Binding is what happens when a method invocation is bound to an implementation
  - Involves lookup of the method in the class, or one or its parents
  - Both method names and parameters are checked
- Binding can happen at two different times
  - Compile time == static binding
  - Run time == dynamic binding


References
Objects
MotorBike bike;
bike = new MotorBike();
MotorBike bike2;
bike2 = bike;
Motor Bike

Static Binding

- References have a type
  - I.e. they refer to instances of a particular Java class
- Objects have a type
  - I.e. they are instances of a particular Java class
  - They are also instances of their super-class
  - Inheritance describes an isA relationship
  - E.g. a MotorBike isA MotorVehicle
- Static binding done by the compiler
  - When it can determine the type of an object
- Method calls are bound to their implementation immediately

Static Binding Example 1

```
//class definition
public class MotorBike {
 public void revEngine() {…}
}

//usage
MotorBike bike = new MotorBike();
motorbike.revEngine();
```

Static Binding Example 2

```
public class MotorVehicle {
  public void start() {…}
  public void stop() {…}
}
public class MotorBike extends MotorVehicle
{
  //overridden
  public void start() {…}
  public void revEngine() {…}
}

//usage. Still statically bound
MotorBike bike = new MotorBike();
motorbike.start();
```

Dynamic Binding

- Achieved at runtime
  - When the class of an object cannot be determined at compile time
  - Means the JVM (not the compiler) must bind a method call to its implementation
- Instances of a sub-class can be treated as if they were an instance of the parent class
  - Therefore the compiler doesn’t know its type, just its base type.

Dynamic Binding Example 1

```
//reference is to base class
MotorVehicle vehicle = new MotorBike();

//method is dynamically bound to
  MotorBike start method
vehicle.start();

//remember all classes derive from Object
Object object = new MotorBike();
object.toString();
```

Dynamic Binding Example 2

```
public interface ElectricalAppliance {
 public void turnOn();
 public void turnOff();
}


public class RemoteControl() {
  public static void
   turnApplianceOn(ElectricalAppliance appliance)
  {
    appliance.turnOn();
  }
}

ElectricalAppliance appliance = …;
RemoteControl.turnApplianceOn(appliance);
```

Dynamic Binding Example 2

```
public class HairDryer implements
   ElectricalAppliance {
}

public class Light implements ElectricalAppliance
{
}

ElectricalAppliance appliance = new HairDryer();
RemoteControl.turnApplianceOn(appliance);

appliance = new Light();
RemoteControl.turnApplianceOn(appliance);
```

References and Behaviours

- The *type of the object* determines its possible behaviours
  - I.e. we define them in the class
- The object reference limits the behaviours we can invoke to those defined by *the type of the reference*

Dynamic Binding Example 3

```
public abstract class HairDryer
  implements ElectricalAppliance {
 //other methods
 public void adjustTemperature();
}

ElectricalAppliance appliance = new
  HairDryer();
//following won't compile
appliance.adjustTemperature();
```

Dynamic Binding Summary

- Whenever a reference refers to an interface or a base class, methods are dynamically bound
- Method implementation determined at runtime
- Dynamic Binding == Polymorphism
  - Very powerful OO feature
- Allows the creation of “frameworks”
  - Applications that are implemented around interfaces, but are customised by plugging in different implementations of those interfaces
  - *Very* extensible

Overview

- Interfaces
- Static Binding
- Dynamic Binding
  - Polymorphism
- Casting

Credits: **http://www.ldodds.com/**

Checking an Objects Type

- Its possible to check the actual type of an object
  - May want to check the real type, if we've only got a reference to an interface or base class
- Use the `instanceof` operator
  - Must be applied to an object, tests whether it has a given type

Instanceof Example

```
//class definition
public class MotorBike implements Vehicle, Motorised
{ … }

//code fragment
Vehicle bike = new MotorBike();

if (bike instanceof MotorBike) {
//do something
}

if (bike instanceof Motorised) {
//do something
}
```

Casting

- If we know the type, we can then "cast" the object

```
Vehicle bike = new MotorBike();
if (bike instanceof MotorBike) {
  MotoBike bike = (MotorBike)bike;
}
```

- If the object isn't of that type, then an exception will be thrown
  - Good idea to always check before casting, unless you're absolutely sure!

references

Credits: **http://www.ldodds.com/**