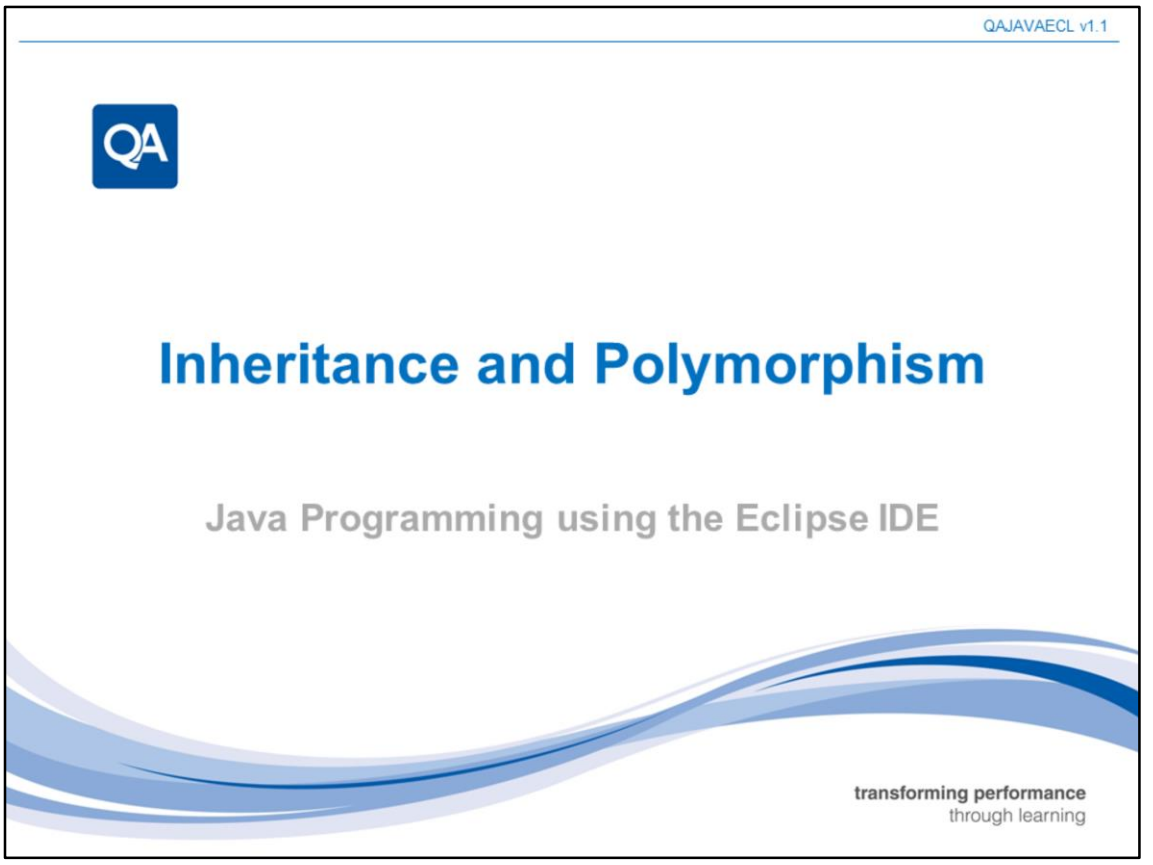

QAJAVAECL v1.1
QA
Inheritance and Polymorphism
Java Programming using the Eclipse IDE
transforming performance
through learning

Outline

- **Inheritance**
  - What is inheritance
  - Inheritance in Java
- **Object hierarchies**
  - abstract
  - extends

Objectives

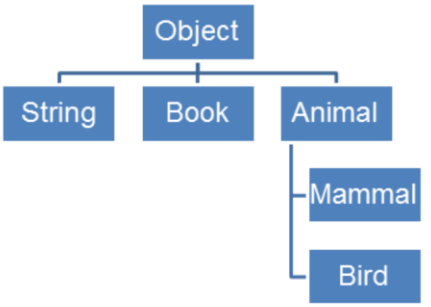
- **By the end of this session we should be able to:**
  - Describe what inheritance is in object oriented programming
  - Be able to use inheritance and abstract classes in Java

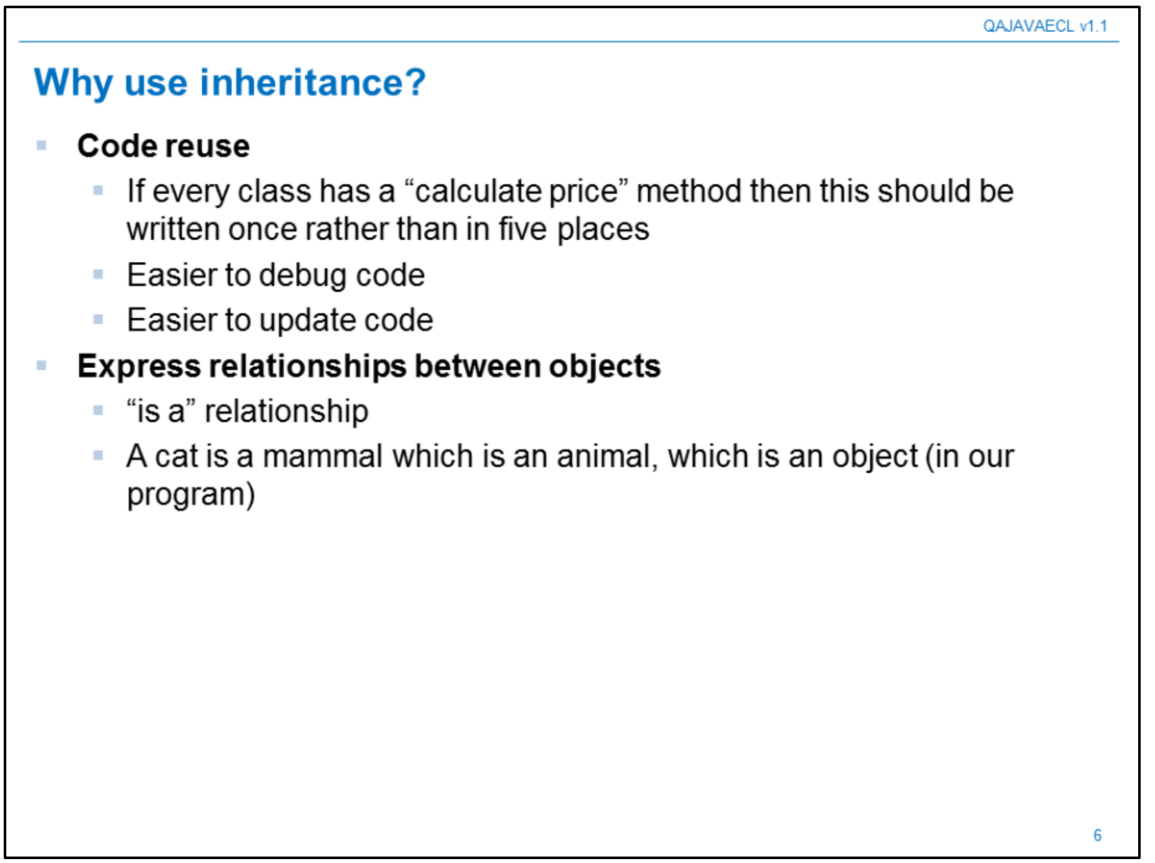
Outline

- **Inheritance**
  - What is inheritance
  - Inheritance in Java
- **Object hierarchies**
  - abstract
  - extends

What is inheritance?

- **Inheritance in the real world is the act of receiving something another**
  - Property, titles, money after people pass away
  - Biological Inheritance - genes passed down through families
- **Inheritance is one of the key concepts in object oriented programming**
  - Objects can inherit fields and methods from other objects
  - This puts everything in an object hierarchy

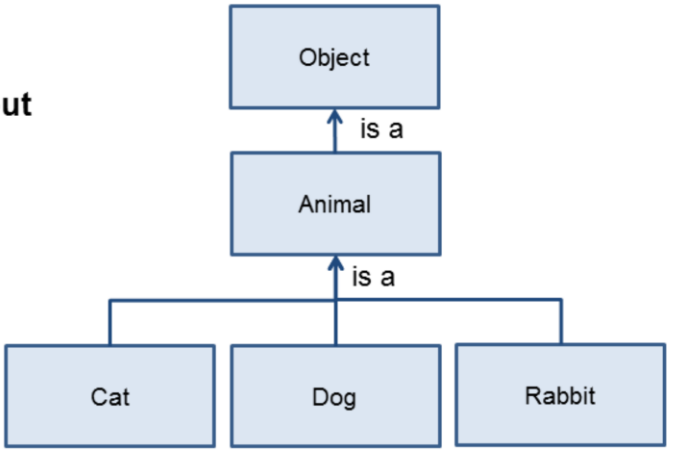




It is also possible to override the behaviour of methods in a hierarchy. The parent object just provides the default assumption for how something will be implemented.

Class hierarchy

- **Using inheritance classes can be structured in a hierarchy**
- **Children can access what is in the parent object**
  - If they are public or protected scope!
- **Parents do not know about theirchildren, so can't access any of their methods**
- **More specific implementations of classes are further down the tree**



Outline

- **Inheritance**
  - What is inheritance
  - Inheritance in Java
- **Object hierarchies**
  - abstract
  - extends

QAJAVAECL v1.1

Extending a class

- To create a sub-class we use the keyword extends
- The sub-class implementation can have additional fields and methods to the parent class

```
public class Animal {
  private String name;
  private int age;

  public Animal
        (String name, int age){
          this.name = name;
          this.age = age;
  }

}
```

```
public class Cat extends Animal {
  boolean fluffy;

  public Cat(String name,
                int age,
                boolean fluffy)
  {
           super(name, age);
           this.fluffy = fluffy;
  }

  public boolean isFluffy(){
          return fluffy;
  }
}
```

9

As Cat is of type Animal to construct the object we need to call Animal's constructor. This has to be the first line of the Cat constructor and pass in the parameters that Animal needs to create an object.

This is then added to by the Cat constructor after the parent has finished.

Inheritance in Java

- **Uses the extends keyword**
  - Only supports single inheritance
  - To access methods in the parent class use the super keyword
- **Every object in Java inherits from the "Object" class**
  - Provides standard methods for all classes
    - toString()
    - hashCode()
    - equals(Object o)
  - The standard implementation of the toString() method returns
    - 'The class name'@'a hash representation of the class'
    - We have been overriding this functionality when declaring our own toString() methods!

Access parent methods and fields

- **Use the keyword super to access a parent object's methods and fields**

In super-class:

```
@Override
public String toString(){
      return "Name: " + name + " Age: " + age;
}
```

In sub-class

```
@Override
public String toString(){
      return "Cat: " + super.toString();
}
```

Public/Protected/Private scope

- **Public**
  - The entire world has access including sub-classes
- **Private**
  - No one outside the class can access the method or field including subclasses
- **Protected**
  - Sub-classes can access this method or field
  - Convention is to use it on methods, and use the accessor methods to access the field values
  - Classes in the same package can access this method or field (even if not sub-classes)

Abstract classes and methods

- **Abstract classes allow for the parent in a relationship to describe what methods should be there without providing an implementation**
  - This class can't be instantiated into an object
  - Uses the keyword abstract in the class declaration
- **Sub-classes then @Override the behaviour of the method in the parent class**

```
public abstract class Shape {
      public abstract double getArea();
}
```

Abstract classes and methods

- **Sub-classes need to provide the method body**
  - If this is not done then the code will not compile
- **@Override notation tells the JVM to use this implementation rather than the one in the parent class**

```
public class Rectangle extends Shape{
	@Override
	public double getArea() {
		return height * width;
	}
}
```

```
public class Circle extends Shape {
	@Override
	public double getArea() {
		return Math.PI * Math.pow(radius, 2);
	}
}
```

Overriding behaviours

- **We can override the behaviour of any method in a super-class this way, not just abstract methods**
  - We've seen this with the toString() method that we are overriding from the Object class

In super-class:

```
@Override
public String toString(){
      return "Name: " + name + " Age: " + age;
}
```

In sub-class

```
@Override
public String toString(){
      return "Cat: " + super.toString();
}
```

Overloading methods

- **What if we want to change the behaviour of a method based on what we pass it?**
  - Two main ways of doing this: overloading and generics
  - (we'll look at generics later)
- **An overloaded method is where the same name is used for a method, but different parameters are used**

```
public String sayHello(){
      return "Hello!";
}

public String sayHello(String name) {
      return "Hello " + name;
}
```

Overloading constructors

- **Overloading is also useful for constructors**

```
public Book(String ID,
            String name,
            String[] authors,
            double price)
            { … }

public Book()
            { /* provide defaults for everything */ }

public Book(String ID,
            String name)
            { /* provide defaults for some fields */ }
```

Outline

- **Inheritance**
  - What is inheritance
  - Inheritance in Java
- **Object hierarchies**
  - abstract
  - extends

Exercise

- Create objects that inherit fields and methods from one another
- Be able to use inheritance and abstract classes in Java

Summary

- **Inheritance**
  - What is inheritance
  - Inheritance in Java
- **Object hierarchies**
  - abstract
  - extends