CM 10227: Lecture 8

Dr Rachid Hourizi and Dr Michael Wright

November 24, 2016

Resources

- More help with this course
  - Moodle
  - E-mail - programming1@lists.bath.ac.uk
- Online C and Java IDE
  - https://www.codechef.com/ide
  - Remember to select Java from the drop down menu.
- Books
  - Java by Dissection (Free pdf online)
  - The Java Tutorial (https://docs.oracle.com/javase/tutorial/)

Resources

- The places that you can get additional support if you are finding the pace of the course a little fast now include
  - A labs (Continued from week 1)
  - B labs
  - ... Wednesday 11:15-13:05 EB0.7
  - ... Fridays 17:15 to 19:15 in CB 5.13)
  - The Drop in Session
  - ... booked 20 min appointments
  - ... Friday 11.15-13.05 1E 3.9
  - PAL sessions (Mondays 14:15 to 15:05 1E 3.9)

This Week

- Interfaces
- Abstract Classes

- Choosing to extend a Class or Abstract Class ...
- ... or implement an Interface

Recap: Interfaces and Abstract Classes

Interfaces

- At some point we may want an even looser relationship between superclass and subclass
- We may want to specify a superclass in which no methods are implemented...
- ... leaving subclasses to do all of the implementation

- A Java Interface is a specification of a type
- ... in the form of a type name and methods
- ... that does not define any implementation of methods

- Interfaces are more properly described as **design patterns**

- Animal interface

```
interface Animal{
   public static final int CONSTANT_VARIABLE = 42;

   String makeSound();

}
```

- Classes implement an interface

```
public class Fox implements Animal{
    public String makeSound(){
        return "Ring-ding-ding-ding-dingeringeding!";
    }
}
```

- NEW : Abstract Classes can also implement an interface

```
abstract class Dog implements Animal {
        public String makeSound(){
            return "woof";
        }

        abstract String breedStandard();
}

  OR

abstract class Dog implements Animal {
        abstract String breedStandard();
}
```

- Important to note that Interfaces are not classes
- ... they can not be instantiated
- ... they cannot share a name with a class
- ... cannot contain implemented methods
- ... can only contain method stubs and constants

- Classes implementing an Interface do not inherit code, but ...
- ... are subtypes of the interface type.
- So, polymorphism is available with interfaces as well as classes.

```
Animal fox = new Fox();

Item item = new MusicFile();
Item item = new VideoFile();
```

- Why does this produce and error?

```
Animal dog = new FrenchBulldog();

String sound = dog.makeSound();
String description = dog.breedDescription(); // ERROR
```

Abstract Classes

- In some situations, however, we may want to include methods in superclasses but only allow them to be used by subclasses
- For example, in the Animal class we might want to include common functionality for all animals
- ... such as average life span

- ... we may also want to include an makeSound() method in the Animal class
- ... but only allow it to be used in classes that describe specific kinds of animal (i.e. in subclasses of Animal)

- In these cases, we can use abstract classes and methods

```java
public abstract class Animal{
   private int averageAge;

   public Animal(int averageAge){
      this.averageAge = averageAge;
   }

   public int getAverageAge(){
      return averageAge;
   }

   // more methods

   // Make the sound of this animal
   abstract String makeSound();
}
```

```java
public class Dog extends Animal{

   private String sound;

   public Dog(int averageAge, String sound){
       super(averageAge);
       this.sound = sound;
   }

   // Make the sound of this animal
   public String makeSound(){
       return sound;
   }

}
```

- NEW : Abstract Classes can also extend Abstract Classes

```
public abstract class Dog extends Animal{
   public static final String SOUND = "woof";

   public Dog(int averageAge){
       super(averageAge);
   }

   // Make the sound of this animal
   public String  makeSound(){
        return sound;
   }

   abstract String breedStandard();

}
```

- NEW : Abstract Classes can also extend Abstract Classes

```
public class BoxerDog extends Dog{

   private String breedStandard;

   public BoxerDog(int averageAge, String sound,
      String breedStandard){
       super(averageAge, sound);
       this.breedStandard = breedStandard;
   }

   public String breedStandard(){
       return breedStandard;
   }
}
```

- Note that some methods can still be written in full in an abstract class ('implemented' methods)
- So an abstract class can have
  - constants
  - fields
  - abstract methods
  - implemented methods

- Also note that, like Interfaces, Abstract Classes can not be instantiated
- Classes implementing an Abstract Class are subtypes of the Abstract Class type.
- So, polymorphism is available with Abstract Class as well as classes.

- Note, however, that Abstract Classes are still classes
- ... more specifically, partially implemented classes

- So the rules that govern implemented classes also govern abstract classes
- i.e. no subclass can have two (or more) abstract parents

Bringing this all together...

- For example,

- ... consider methods of payment accepted in your application
- ... we know we need to take payment but there are different ways of paying
- ... e.g. Credit Card, PayPal etc..

```java
interface Payment{
    void makePayment(double debit);
}
```

```java
public class PayPal implements Payment{
    public void makePayment(double debit) {
        // logic for PayPal payment
        // e.g. Paypal uses username and password for
           payment
    }
}
public class CreditCard implements Payment{
    public void makePayment(double debit){
        // logic for CreditCard payment
        // e.g. CreditCard uses card number, date of
           expiry etc...
    }
}
```

Bringing this all together...

- BUT, if for example...
- ... each payment type (PayPal, Credit Card etc.) needs to be authorised by a Bank
- ... and this process is the same for all types of payment
- ... and we don't want programmers to reimplement this each time

- ... we would use an Abstract Class to implement this method
- ... but ensure that programmers implement the makePayment method
- ... because this is specific to each payment type

```
public abstract class Payment{

    protected boolean authoriseWithBank(String
        userAuthKey){
         // logic to authorise payment with bank
    }

    abstract void makePayment(double debit);
}
```

```java
public class PayPal extends Payment{
    public void makePayment(double debit) {
        // logic for PayPal payment
        // e.g. Paypal uses username and password for
            payment
        super.authoriseWithBank(authToken);
    }
}
public class CreditCard extends Payment{
    public void makePayment(double debit){
        // logic for CreditCard payment
        // e.g. CreditCard uses card number, date of
            expiry etc...
        super.authoriseWithBank(authToken);
    }
}
```

Bringing this all together...

- What's important is the following code does not care if Payment is
- ... an Interface
- ... or an Abstract Class

- ... because we refer to it by its supertype
- ... we know the method(s) we can call on this supertype (specified in the interface or abstract class)
- ... and therefore implemented (or inherited) in all of its subtypes

```java
public class NozamaUserDetails{

    private String name;
    private Payment paymentMethod;

    public NozamaUserDetails(String name, Payment
        paymentMethod){
         this.name = name;
         this.paymentMethod = paymentMethod;
    }

    // other methods...
}
```

```
public class NozamaRegisterUser{

   public NozamaUserDetails registerUser(String name){
        Payment p = getPaymentMethod();
        return new NozamaUserDetails(name,
           paymentMethod);
    }

    private Payment getPaymentMethod(){
        return new PayPal();
    }
}
```

```java
public class ProcessPayment{

    public void purchase(NozamaUserDetails user, Item
        item){
         Payment p = user.getPaymentMethod();
         p.makePayment(item.getPrice());
    }

    // other methods...
}
```

```java
public class Ping implements Payment{
    public void makePayment(double debit) {
        // logic for Ping payment
        // e.g. mobile phone number
        // ... and authorising with bank ??
    }
}

public class Ping extends Payment{
    public void makePayment(double debit) {
        // logic for Ping payment
        // e.g. mobile phone number
        super.authoriseWithBank(authToken);
    }
}
```

Interfaces vs Abstract Classes

- It can be difficult to identify
  - when to use an abstract class
  - when to use an interface
- As a simple rule of thumb, when faced with a choice between abstract classes and interfaces
  - use an abstract class when
  - you want to implement some but not all of a class's methods
  - and you are willing to accept the restrictions imposed upon classes
  - e.g. single inheritance
  - otherwise use an interface

Choosing to extend a Class or Abstract Class or Implement an Interface

Interfaces Over Abstract Classes or Classes

- Want to include methods in superclasses but only allow them to be used by subclasses
- Know what we want but not how to do it

Abstract Classes Over Interfaces

- We know some of want we want to do (i.e. can provide common functionality) ...
- ... but not all

Classes Over Abstract Classes

- When would we choose to override methods rather than declaring them abstract?
- ... lets think about the DOME example
- ... real world, physical things on a Book Shelf
- ... and a Database to keep track of my stuff

Summary

- Inheritance can provide shared implementation...
- ... both as concrete and abstract classes
- Inheritance provides shared type information
- ... interfaces

Summary

- Abstract classes function as incomplete superclasses
- Interfaces provide specification without implementation

- Both Interfaces and Abstract Classes support polymorphism