Questions and Exercises

These questions and exercises is an opportunity to see what you've learnt from the lecture as well as practice the new things we've been talking about. In other words, these questions and exercises are completely optional but it's recomended to do them. In the end of the document you will find the answers to the questions as well as possible solutions to the exercises, note that one can solve an exercise in different ways. There will also be some suggestions about what one could code if one want to continue with some more advanced things. These suggestions will not come with a possible solution and might include things that haven't been covered in the lecture.

Question 1
What does it mean when one class is extending another?

Question 2
How is the super keyword used when calling constructors and methods of superclasses?

Question 3
How does one override a method? And what does it mean and do?

Question 4
What does the final keywork do when used on fields, methods and classes?

Question 5
What is an anynomous class? And is there any reason to use it?

Question 6
What's the difference between the protected and the default scope?

Question 7
When one overrides a method one can add @Override just before it. What is this good for?

Question 8
What does instanceof do?

Exercise 1
Write a subclass to the Random class. This subclass should contain an extra method called nextString. It should take one ineteger parameter. The string that should be returned is a randomized string with the length given by the parameter. Each character should be randomized from the

english letters and numbers.

Exercise 2
Write the class shoe that consists of the name and price of the shoe and some methods. Write two subclasses, one platform shoe and one high-heeled one. Write a simple person class where the person can wear a type of shoe. Also give the person a height and name. Now use this to allow the user to go to different fun-park rides which has a height limit, the platfrom shoe should increase the total height of the user while the high-heeled one should increase it even further but however give a chance of you falling over in approaching the ride.

Further explorations
Use the code shown in the lecture(can be found on the lecture page) and implement the doors shown in the image on the lecture page (Annoying door, Breakable door and Code door).

When that is done, implement windows(one way doors to jump out of), finger print doors(which make sure that it's the correct person trying to unlock it), counting doorway (keeps count how many times somebody has gone through it).

Answers and solutions

Answer to Question 1
When a class (the subclass) extends another class (the superclass) it means that the subclass will inherit everything from the superclass. In other words this means that all the fields and methods in the superclass also can be used in the subclass(given that their scopes allow it).

Since a subclass is an extention of the superclass an object that has been created from the subclass can be stored in a variable with the superclass as its type.

Answer to Question 2
In the beginning of a constructor, one constructor of the superclass must be called. This is done by calling super, it could be something like super(3, true) if one of the constructors of the superclasses requires an integer and a boolean.

When refering to a method of a superclass, this can be done from anywhere. To do so, simply use super.methodcall. So it could look something like: super.myMethod(3,2);

Answer to Question 3
A method can be overridden in a subclass if that subclass contains a method with the same name and parameter list as the method to be overridden. If a method is overriding another method that method will run instead when being called by an instance of the subclass.

Answer to Question 4

The final keyword can be used on methods to prevent them from being overriden in a subclass. When used on a class however, it prevents any classes from extending the class itself.

When used on a field it simply prevents you from changin the value of the field. Furtheremore, the field can only be initialized in the constructor.

Answer to Question 5
An anonymous class is a class without a name. These are created at the same time as you create the instance of the class. This means that it can be handy to use them if you only need a class once which is not too complex. If you want to use a class multiple times or if your class contains a lot of things then it's better to just use a normal class. The example belo shows you an example of a anonymous class:

```
new Doorway(hallway, elevator) {
        @Override
        public void goThrough(Person person) {
                System.out.println("You can't get through here");
        }

        @Override
        public String getLabel(Room room) {
                return room.getName() + " [out of service]";
        }
};
```

Answer to Question 6
The default scope only allow you to access the field or the method from within the same package. However, the protected scope allow you to get access from within the same package and from any subclasses, no mather from which package they come from.

Answer to Question 7
By adding @Override in front of a method forces us to use the method to override a method in the superclass. This means that if we messed up our method (maybe by typing it wrong or giving it the wrong parameter list) we will get a compilation error that tells us that we're not overriding a method.

Note that @Override is not necessary but is highly recommended.

Answer to Question 8
If you have an object you can use instanceof to check if it is of the correct type. So an object created from the given class will return true when checked if it's an instance of that class. However, an object created from a subclass of the given class will also return true when checked. This means that you can check if a given object contains the fields and methods from a given class.

Possible solution to Exercise 1

```
import java.util.Random;

public class ExtendedRandom extends Random {

   public ExtendedRandom() {
      super();
   }

   //constructor for the seed version of Random (optional)
   public ExtendedRandom(long seed) {
      super(seed);
   }

   //the characters we want to randomzie from
   private static final String validChars =
 "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";

   public String nextString(int length) {
      String randomString = "";

      for (int i = 0; i < length; i++) {
         //add another character
         randomString += validChars.charAt(nextInt(validChars.length()));
      }

      return randomString;
   }

}
```

Possible solution to Exercise2

```
public class Exercise2 {
   public static void main(String[] args) {
      Ride[] rides = {
         new Ride("Ride of Doom!", 160),
         new Ride("Awesomesauce", 150),
         new Ride("Ride among the goats", 145),
         new Ride("Duckride", 140)
      };

      Shoe[] shoes = {
         new Shoe("Plain shoes", 30),
         new PlatformShoe("Mud under my shoes", 20),
         new HighHeeledShoe("Fancy shoes for the wrong occation", 100)
      };
```

```
        Person steve = new Person("Steve", 140);

        for (Shoe shoe : shoes) {
            steve.setShoes(shoe);

            System.out.println(shoe.getName());
            for (Ride ride : rides) {
                System.out.print(ride.getName() + ": ");
                ride.enjoyRide(steve);
            }
            System.out.println();
        }
    }
}

public class Person {

    private String name;
    private int height;
    private Shoe shoes;

    public Person(String name, int height) {
        this.name = name;
        this.height = height;
    }

    public void setShoes(Shoe shoes) {
        this.shoes = shoes;
    }

    public int getTotalHeight() {
        if (shoes != null) {
            return height + shoes.getHeight();
        }else{
            return height;
        }
    }

    //returns true if the person got to the ride
    public boolean moveToRide() {
        if (shoes != null) {
            return shoes.moveToRide();
        }else{
            return true;
        }
    }
}
```

```
public class Ride {

    private String name;
    private int heightLimit;

    public Ride(String name, int height) {
        this.name = name;
        heightLimit = height;
    }

    public void enjoyRide(Person person) {
        if (person.moveToRide()) {
            if (person.getTotalHeight() >= heightLimit) {
                System.out.println("Ooooooooh, what a ride!");
            }else{
                System.out.println("I'm sorry, you're too short for this ride");
            }
        }
    }

    public String getName() {
        return name;
    }
}


public class Shoe {
    private String name;
    private int price;

    public Shoe(String name, int price) {
        this.name = name;
        this.price = price;
    }

    public String getName() {
        return name;
    }

    public int getPrice() {
        return price;
    }

    public int getHeight() {
        return 0;
    }

    public boolean moveToRide() {
```

```
        return true;
    }

}

public class PlatformShoe extends Shoe {

    public PlatformShoe(String name, int price) {
        super(name, price);
    }

    @Override
    public int getHeight() {
        return 5;
    }
}

import java.util.Random;
public class HighHeeledShoe extends Shoe {

    private Random myRandomGenerator;

    public HighHeeledShoe(String name, int price) {
        super(name, price);
        myRandomGenerator = new Random();
    }

    @Override
    public int getHeight() {
        return 10;
    }

    @Override
    public boolean moveToRide() {
        if (myRandomGenerator.nextInt(10) < 3) {
            System.out.println("You tripped on the way to the ride, you should really learn to walk in
these shoes");
            return false;
        }else{
            return true;
        }
    }
}
```