COMP503/ENSE502/ENSE602: Week 5 – Exercises

➢ **Exercise 1:** The Animal Hierarchy

Create the Animal class with the following definition:

```
public class Animal {
	protected int eat;
	protected int happy;

	public Animal(){
		this.eat = 0;
		this.happy = 0;
	}

	public void feed(){

	}

	public String toString(){
		return "Eaten: "+this.eat+" Happy: "+this.happy;
	}

}
```

Create two subclasses of Animal using the **extends** keyword. The first subclass is the **Cat** class which is defined as follows:

```
public class Cat extends Animal{
	public void feed(){
		this.eat+=1;
		this.happy+=10;
	}
	public Cat(){
		System.out.println("Cat object instantiated!");
		this.happy = -10;
	}
	public String toString(){
		return "Cat has "+super.toString();
	}

}
```

Step through your code to determine what **super** does in the **toString** method.

a) Use the **Cat** class as an example to define the **Dog** class that extends **Animal** such that

- The **feed** method increments the **eat** instance variable by 5 and **happy** by 1.
- The default constructor prints "**Dog** object instantiated!"
- The **toString** method prints a suitable string.

b) Create the **AnimalApplication** class, which maintains an array of **Animal** objects (with a fixed size of 4). The application allows the user to instantiate either **Dog** or **Cat** objects according to what they input in the menu, and these objects are stored in the array. Once the objects are created, the array is printed to the user. The user select an **Animal** object from the list to feed. Your application should produce similar console output as represented in the next page.

Sample output:

```
Type 1 to create a Dog Object or 2 to create a Cat object
2
Cat object instantiated!

Type 1 to create a Dog Object or 2 to create a Cat object
1
Dog object instantiated!

Type 1 to create a Dog Object or 2 to create a Cat object
2
Cat object instantiated!

Type 1 to create a Dog Object or 2 to create a Cat object
1
Dog object instantiated!

Select an animal to feed by entering a number within the range: 0 to 4
0 Cat has Eaten: 0 Happy: -10
1 Dog has Eaten: 0 Happy: 10
2 Cat has Eaten: 0 Happy: -10
3 Dog has Eaten: 0 Happy: 10
0

Select an animal to feed by entering a number within the range: 0 to 4
0 Cat has Eaten: 1 Happy: 0
1 Dog has Eaten: 0 Happy: 10
2 Cat has Eaten: 0 Happy: -10
3 Dog has Eaten: 0 Happy: 10
3

Select an animal to feed by entering a number within the range: 0 to 4
0 Cat has Eaten: 1 Happy: 0
1 Dog has Eaten: 0 Happy: 10
2 Cat has Eaten: 0 Happy: -10
3 Dog has Eaten: 5 Happy: 11
2

Select an animal to feed by entering a number within the range: 0 to 4
0 Cat has Eaten: 1 Happy: 0
1 Dog has Eaten: 0 Happy: 10
2 Cat has Eaten: 1 Happy: 0
3 Dog has Eaten: 5 Happy: 11
4
Input out of range, quitting.
```

COMP503/ENSE502/ENSE602: Week 5 – Exercises

➢ **Exercise 2:** The Shapes Hierarchy

Create the following **Shape** class:

```
public class Shape {

      public double computeArea(){
            return 0.0;
      }
}
```

Derive two subclasses of **Shape**: **Rectangle** and **Circle**.

a) In both the **Rectangle** and **Circle** class,
   - Declare appropriate instance variables for each shape to compute its area (e.g. length, width, radius)
   - Define a constructor with appropriate input parameters to initialise the instance variables
   - Override the **computeArea** method to compute the area appropriate to the shape.

b) Create the class **ShapeApp** with a **main** method, which prints a menu asking the user to either create a **Circle** or **Rectangle** object. You can either fix the size of these shapes or use a random number generator (e.g. using the **Random** class) to generate random shape dimensions. Your console output should look similar to the following:

Sample output:

```

1 Create a Circle Object
2 Create a Rectangle object
3 Stop
1
The area of the circle is: 3.141592653589793

1 Create a Circle Object
2 Create a Rectangle object
3 Stop
2
The area of the rectangle is: 4.0

1 Create a Circle Object
2 Create a Rectangle object
3 Stop
3
```

Note that we never need to instantiate the **Shape** class. It doesn't even have any information to compute the area and so the **computeArea** method returns the misleading value of 0.0.

c) This exercise introduces the concept of an abstract class. Review this week's lecture slides for details. Change the **Shape** class to an abstract class by adding the **abstract** keyword to the class declaration: **abstract public class Shape**. Make the method **computeArea** abstract as well by using the **abstract** keyword again in front of the **public** access modifier. Now, we do not need to write a body for this method.

➢ **Exercise 3:** The **SparkPlug** Mobile Phone Provider

The telecommunication packages offered by the New Zealand mobile phone provider **SparkPlug** offer customers varying amounts of talk time, texts and data for different prices.

a) Create Java classes that model each of the following mobile packages. Remember, it is up to you to optimise your code re-use.

- The **Standard** package provides a mobile service keeping track of how many *minutes the customer talks* and the *number of texts sent*.
    - There are two functionalities for talking and texting, modelled by methods
        - **public void talk(int nMinsTalked)**
        - **public void sendTexts(int nTextSent)**

        - These methods check if talk or text limits are reached. If so, the customer is notified via a message (printed to the console) whenever they try to talk or send texts. Limits are specified when a Standard object is created.
- **Data** mobile packages have the same functionality as **Standard**, but also have *data usage (and limits)*. There are two kinds of Data packages: **Heavy** and **Lite**. Both of these packages offer data transfer functionality, modelled by the method

  public void transfer(int n)

  which checks data limits, notifying the customer when their limit is reached (hint: the ***transfer*** method should be in **Data** class as parent class and inherited by **Heavy** and **Lite** classes).

b) Create methods to compute the cost of talking, texting and data, appropriate to the package.

- Standard package: Talk: 10¢/min, text: 5¢/text
- Data Lite package: Talk: 10¢/min, text: 5¢/text, Data: 15¢/MB
- Data Heavy package: Talk: 10¢/min, text: free, Data: 25¢/MB

c) Create the **SparkPlugApp** class with a main method that creates an array of mobile package objects and simulates the use of the various package functionality, calling the methods: **sendText**, **talk** and **transfer**.

Sample output:

```
Please select a mobile phone package.
0: Talk: [0/10] Text: [0/100]
1: Talk: [0/20] Text: [0/56] Data: [0/1024]
2: Talk: [0/30] Text: [0/156] Data: [0/5120]
3: stop.
1
Talk: [0/20] Text: [0/56] Data: [0/1024]
1. Talk 2. Text 3. Data 4. Choose another plan
1
Talk: [16/20] Text: [0/56] Data: [0/1024] Amount owing on account is: $1.60
1. Talk 2. Text 3. Data 4. Choose another plan
1
Talk: [36/20] Text: [0/56] Data: [0/1024] Amount owing on account is: $3.60
1. Talk 2. Text 3. Data 4. Choose another plan
4
Talk: [36/20] Text: [0/56] Data: [0/1024] Amount owing on account is: $3.60
0: Talk: [0/10] Text: [0/100]
1: Talk: [36/20] Text: [0/56] Data: [0/1024]
2: Talk: [0/30] Text: [0/156] Data: [0/5120]
3: stop.
2
Talk: [0/30] Text: [0/156] Data: [0/5120]
1. Talk 2. Text 3. Data 4. Choose another plan
3
Talk: [0/30] Text: [0/156] Data: [301/5120] Amount owing on account is:
$75.25
1. Talk 2. Text 3. Data 4. Choose another plan
3
Talk: [0/30] Text: [0/156] Data: [648/5120] Amount owing on account is:
$162.00
1. Talk 2. Text 3. Data 4. Choose another plan
4
Talk: [0/30] Text: [0/156] Data: [648/5120] Amount owing on account is:
$162.00
0: Talk: [0/10] Text: [0/100]
1: Talk: [36/20] Text: [0/56] Data: [0/1024]
2: Talk: [0/30] Text: [0/156] Data: [648/5120]
3: stop.
3
```