**Week 1 Tutorial – Tutor Instructions and additional solutions**

**Students require access to the following software during the term:**

* NetBeans 8.2
* Java 8(Oracle)
* JavaFX SceneBuilder 8.5.0

(see the resources block on the right-hand side of the unit website for links to instructions about how to download the software on personal computers.)

**If classes are on zoom: Please remind students to make sure they have access to all the software required in their tutorial classes. Over the next two weeks prepare them for the weekly submissions required in weeks 3-12 that are for marks. They must have the software needed in each tutorial**

**If classes are face to face on campus: The software should be installed in the campus labs. Please check that the software is available in the lab you will have your class. Check that the week 4 instructions work for JavaFX development in your lab. (It is possible that the version of SceneBuilder we are using is not at the default location.)**

This week students need access to Java, NetBeans. They should also install and become familiar with Violet to draw their UML diagrams if/when required. Note that there is also a link to document that tells students where to view a YouTube video that will help them to get started with the NetBeans debugger.

**Aim in week 1**

* **Question 1**
* **If possible have the students code the singleton example (question 2). If insufficient time for students to code it, try to leave enough time to go through the solution to this question in class and explain how it works. If students do not code it in class, tell them to complete this coding exercise for homework**
* **I’d expect questions 3, 4 and 5 to be left as homework exercises.**

**Question 1 (simple revision to get started)**

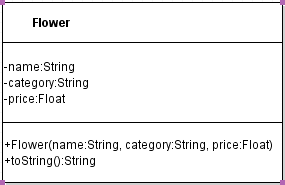
**Part (a) and (b)**

Students can initially draw their UML diagrams by hand in the class.

However, please introduce students to **Violet drawing tool** so that students can learn how to use the tool to draw their UML diagrams in any documentation required in the future (see the link in the resource block to download Violet). Violet was used to produce the diagrams in the following solutions.

**(Activity 1 in lecture slides)**

**Students were asked to produce the following UML as part of an activity in the week 1 lecture:**



**And now in the tutorial question students are asked to create program with this class and a driver class to test their Flower class.**

The exercise will be discussed in the lecture. Students should code the Flower and “driver” class using NetBeans as their first practical exercise in the tutorial

**In class have students use NetBeans to create a program with the “driver” class (containing main) and a Flower class. Create a Flower and print out the information about the flower using the flower’s toString method. Student code should be similar to the following:**

**public class Flower1 {**

**/\*\***

**\* @param args the command line arguments**

**\*/**

**public static void main(String[] args) {**

**Flower f = new Flower("daisy","filler",(float)2.00 );**

**System.out.println(f);**

**}**

**}**

**public class Flower {**

**private String name;**

**private String category;**

**private float price;**

**public Flower(String n, String c, float p){**

**name = n;**

**category = c;**

**price = p;**

**}**

**@Override**

**public String toString(){**

**String p = String.format("$%.2f",price);**

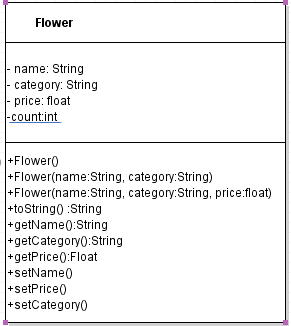
**return "< "+name+" "+category+" "+p+">";**

**}**

**}**

**Part (c) and (d)**

Students are asked implement the UML diagram created in Activity 2 in the lecture. They are to add a class variable (static count) to store the number of flowers created and test their implementation of Flower by creating four flowers. Each flower is to be created using a different constructor. Students are to display the information about each flower and also a message that displays the value of count (i.e. the number of flowers have constructed on three different occasions).



\*Add

**+getCount():int**

In the methods in the above diagram when you go though the solution – it is missing in the diagram

**The FlowerTest.java code (main method)**

public class FlowerTest {

public static void main(String[] args) {

System.out.printf("The number of flowers is now %d%n",

Flower.getCount());

Flower f = new Flower("daisy","filler",(float)2.00 );

System.out.println(f);

Flower f2 = new Flower();

System.out.println(f2);

System.out.printf("The number of flowers is now %d%n",

f.getCount());

Flower f3 = new Flower("rose","focal");

System.out.println(f3);

Flower f4 = new Flower("snap dragon");

System.out.println(f4);

System.out.printf("The number of flowers is now %d%n",

Flower.getCount());

}

}

**The Flower class (Flower.java)**

public class Flower {

private String name;

private String category;

private float price;

private static int count;

public Flower(){

name= "unknown";

category = "unknown";

count++;

}

public Flower(String n, String c, float p){

setName(n);

category = c;

price = p;

count++;

}

public Flower(String n, String c){

setName(n);

category = c;

count++;

}

public Flower(String n){

setName(n);

category = "unknown";

count++;

}

@Override

public String toString(){

String p = String.format("$%.2f", getPrice());

return "< "+getName()+" "+getCategory()+" "+p+">";

}

/\*\*

\* @return the name

\*/

public String getName() {

return name;

}

/\*\*

\* @param name the name to set

\*/

public void setName(String name) {

this.name = name;

}

/\*\*

\* @return the category

\*/

public String getCategory() {

return category;

}

public static int getCount() {

return count;

}

/\*\*

\* @param category the category to set

\*/

public void setCategory(String category) {

this.category = category;

}

/\*\*

\* @return the price

\*/

public float getPrice() {

return price;

}

/\*\*

\* @param price the price to set

\*/

public void setPrice(float price) {

this.price = price;

}

}

**Question 2 – demonstration of singleton pattern**

**In singleObject.java:**

public class SingleObject {

//create an object of SingleObject

String name;

private static SingleObject instance = null;

//make the constructor private so that this class cannot be

//instantiated

private SingleObject(String n){

name = n;

}

//Get the only object available

public static SingleObject getInstance(String n){

if (instance == null) {

instance = new SingleObject(n);

}

return instance;

}

public void setName(String n) {

name = n;

}

public void showMessage(){

System.out.println(name+ " says Hello World!");

}

**}**

**In SingletonPatternDemo:**

public class SingletonPatternDemo {

public static void main(String[] args) {

//Get the only object available

SingleObject object1 = SingleObject.getInstance("Mairi");

//show the message

object1.showMessage()

}

**}**

**Answers to the questions**

1. getInstance() is static as you need to be able to invoke it even when no object exists
2. SingleObject instance is static – again you need to be able to access it even i=f there is no instance of the class – it gets accessed through the static method getInstance()
3. The constructo is private so that it can’t be used outside the class to create new instances – we don’t want to allow the creation of more than one instance (singleton)

**Questions 3, 4 and 5**

Code for the solutions to questions 3, 4, and 5 can be found in the solutions in week 1.

Note that

1. the comments need to be changed – currently they do not reflect the correct tutorial question number.)
2. The current solutions use a GUI. However, we don’t study JavaFX until week 4 and we don’t study Swing. I’d get the students to use print statements to prompt for the input and use a Scanner to read the input in these exercises.