Lab3

March 6, 2024

1 Lab 3

Deadline: **Week 4** in your respective lab session

1.0.1 Name:Hansel Rodrigues

1.0.2 Student ID:230603866

1.1 Question 1 [1 mark]

Write class `Utils`, which will contain useful constants and methods. It should contain a `final` class variable `PI` and store a value `3.14`. The class `Utils` should also contain two class methods, `circlePerimeter` and `circleArea`.

`circlePerimeter` should take the circle's radius as an argument, calculate and return the circle's perimeter with such radius with exactly 1 decimal place rounded down.

The formula for the perimeter of a circle is:
$circle\ \ perimeter = 2\pi r$ where r is a radius.

`circleArea` should take the circle's radius as an argument, calculate and return the area of the circle with such radius with exactly 1 decimal place rounded down.

The formula for the area of a circle is:
$circle\ \ area = \pi r^2$ where r is a radius.

You are NOT allowed to use libraries or modify a String representation of the number to achieve this.

Lastly, write class `Main1` with the main method that asks the user to input the circle's radius and prints out the value of , the circle perimeter and the area with such radius. To achieve this, you must use a variable and methods defined within `Utils`.

Example run:

```
What is the radius of your circle? 26
The value of  is 3.14
The perimeter of your circle is 163.2 units
The area of your circle is 2122.6 square units
```

Write your answer below:

[2]:
```
class Utils
{
    static final double PI = 3.14;

    public static double circlePerimeter(double radius)
    {
        return (double)((int)(2 * PI * radius * 10)) / 10;
    }

    public static double circleArea(double radius)
    {
        return (double)((int)(PI * radius * radius * 10)) / 10;
    }
}
```

[3]:
```
class Main1
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        System.out.print("What is the radius of your circle? ");
        double radius = scanner.nextDouble();
        System.out.println("The value of   is " + Utils.PI);
        double perimeter = Utils.circlePerimeter(radius);
        System.out.println("The perimeter of your circle is " + perimeter + "␣
 ↪units");
        double area = Utils.circleArea(radius);
        System.out.println("The area of your circle is " + area + " square␣
 ↪units");
    }
}
```

Run your program:

[4]:
```
Main1.main(null)
```

```
What is the radius of your circle?
 26
The value of   is 3.14
The perimeter of your circle is 163.2 units
The area of your circle is 2122.6 square units
```

1.2 Question 2 [1 mark]

Write a class `Dictionary` which can store up to `10000` words and their definitions.

The class `Dictionary` has two instance methods `addEntry` and `findDefition`.

`addEntry` takes two Strings as arguments, a word and its definitions and adds them to the array with all the other entries.

`findDefinition` takes a String as an argument, a word, and checks whether it was entered in the dictionary. If found, it returns its definitions; otherwise, it returns `"Not Found."`.

Solutions that concatenate a word and its definition into 1 String to store them inside the array will not be accepted. We are looking for an Object-Oriented solution.

Hint: Write another class, `DictionaryEntry`.

Write your answer below:

[6]:
```
class DictionaryEntry
{
    String word;
    String definition;

    public DictionaryEntry(String word, String definition)
    {
        this.word = word;
        this.definition = definition;
    }

    public String getWord()
    {
        return word;
    }

    public String getDefinition()
    {
        return definition;
    }
}
```

[7]:
```
class Dictionary
{
    static final int MAX_ENTRIES = 10000;
    DictionaryEntry[] entries;
    int size;

    public Dictionary()
    {
        entries = new DictionaryEntry[MAX_ENTRIES];
        size = 0;
    }

    public void addEntry(String word, String definition)
```

```
    {
        if (size < MAX_ENTRIES)
        {
            entries[size] = new DictionaryEntry(word, definition);
            size++;
        }
        else
        {
            System.out.println("Dictionary is full. Cannot add more entries.");
        }
    }

    public String findDefinition(String word)
    {
        for (int i = 0; i < size; i++)
        {
            if (entries[i].getWord().equals(word))
            {
                return entries[i].getDefinition();
            }
        }
        return "Not Found.";
    }
}
```

[8]:
```
class Main2 {
    public static void main(String[] args) {
        Dictionary englishLanguage = new Dictionary();

        englishLanguage.addEntry("dog", "A loyal mammal, domesticated for␣
 ↪companionship, belonging to the Canidae family.");
        englishLanguage.addEntry("cat", "A cat is a domesticated feline mammal,␣
 ↪valued for companionship, belonging to the Felidae family.");

        System.out.println(englishLanguage.findDefinition("dog"));
        System.out.println(englishLanguage.findDefinition("cat"));
        System.out.println(englishLanguage.findDefinition("octopus"));
    }
}
```

Run your program:

[9]:
```
Main2.main(null);
```

```
A loyal mammal, domesticated for companionship, belonging to the Canidae family.
A cat is a domesticated feline mammal, valued for companionship, belonging to
the Felidae family.
```

```
Not Found.
```

1.3 Question 3 [1 mark]

Paste your code from Question 2 below and modify it (start by renaming each class) to implement the Singleton Design Pattern so that you can create only one instance of a class `Dictionary`. If you attempt to create a second instance of a `Dictionary`, print out an error message `"This class is a singleton!"` and return `null`.

Singleton is a design pattern in software engineering. It is used to ensure there is only one instance of a particular class. It prevents us from accidentally creating more instances than we want to have.

For example, if we are developing a system to manage books in the library, we would only want one instance to keep track of all the books. Accidentally creating multiple instances storing data about all the books would (unintentionally) lead to numerous inconsistencies.

Write your answer below:

[10]:
```
class DictionaryEntry
{
    String word;
    String definition;

    public DictionaryEntry(String word, String definition)
    {
        this.word = word;
        this.definition = definition;
    }

    public String getWord()
    {
        return word;
    }

    public String getDefinition()
    {
        return definition;
    }
}
```

[11]:
```
class Dictionary2
{
    static final int MAX_ENTRIES = 10000;
    private static Dictionary2 instance;
    DictionaryEntry[] entries;
    int size;

    private Dictionary2()
```

```
    {
        entries = new DictionaryEntry[MAX_ENTRIES];
        size = 0;
    }

    public static Dictionary2 createDictionary2()
    {
        if (instance == null)
        {
            instance = new Dictionary2();
        }
        else
        {
            System.out.println("This class is a singleton!");
            return null;
        }
        return instance;
    }

    public void addEntry(String word, String definition)
    {
        if (size < MAX_ENTRIES)
        {
            entries[size] = new DictionaryEntry(word, definition);
            size++;
        }
        else
        {
            System.out.println("Dictionary is full. Cannot add more entries.");
        }
    }

    public String findDefinition(String word)
    {
        for (int i = 0; i < size; i++)
        {
            if (entries[i].getWord().equals(word))
            {
                return entries[i].getDefinition();
            }
        }
        return "Not Found.";
    }
}
```

[12]:
```
class Main3 {
    public static void main(String[] args) {
```

```
        Dictionary2 englishLanguage = Dictionary2.createDictionary2();
        Dictionary2 welshLanguage = Dictionary2.createDictionary2();

        englishLanguage.addEntry("dog", "A loyal mammal, domesticated for␣
 ↪companionship, belonging to the Canidae family.");
        englishLanguage.addEntry("cat", "A cat is a domesticated feline mammal,␣
 ↪valued for companionship, belonging to the Felidae family.");

        System.out.println(englishLanguage.findDefinition("dog"));
        System.out.println(englishLanguage.findDefinition("cat"));
        System.out.println(englishLanguage.findDefinition("octopus"));

        System.out.println(welshLanguage); // null
    }
}
```

Run your program:

[13]:
```
Main3.main(null);
```

```
This class is a singleton!
A loyal mammal, domesticated for companionship, belonging to the Canidae family.
A cat is a domesticated feline mammal, valued for companionship, belonging to
the Felidae family.
Not Found.
null
```

1.4 Question 4 [1 mark]

Write class `Animal`, which stores the animal's name and age. It also contains a constructor and a method `displayInfo` that prints out information about the animal. The class `Animal` has 2 sub-classes `Dog` and `Cat`.

`Dog` has an instance variable `bread`, which is initialised through its constructor. It also has a method `bark` that prints out a bark sound and the dog's breed.

`Cat` has an instance variable `hasClaws` which is initialised through its constructor. It also has a method `meow` that prints out a meow sound and whether a cat has claws.

Lastly, define class `Main4` to test your code.

Write your answer below:

[14]:
```
class Animal
{
    String name;
    int age;
```

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

    public void displayInfo()
    {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}
```

[15]:
```
class Dog extends Animal
{
    private String bread;

    public Dog(String name, int age, String bread)
    {
        super(name, age);
        this.bread = bread;
    }

    public void bark()
    {
        System.out.println("Woof! I am a " + bread);
    }
}
```

[16]:
```
class Cat extends Animal
{
    private boolean hasClaws;

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

    public void meow() {
        System.out.println("Meow! I " + (hasClaws ? "have" : "don't have") + "␣
 ↪claws");
    }
}
```

[17]:
```
class Main4 {
    public static void main(String[] args) {
        Dog myDog = new Dog("Buddy", 3, "Golden Retriever");
```

```
        myDog.displayInfo();
        myDog.bark();

        System.out.println();

        Cat myCat = new Cat("Whiskers", 2, true);
        myCat.displayInfo();
        myCat.meow();
    }
}
```

Run your program:

[18]:
```
Main4.main(null);
```

```
Name: Buddy
Age: 3
Woof! I am a Golden Retriever

Name: Whiskers
Age: 2
Meow! I have claws
```

1.5 Question 5 [1 mark]

You and two of your friends are arguing about the following probability puzzle:

You are given a black box containing 100 balls, n of them are red, and $100 - n$ are green, where n is an integer that is chosen uniformly at random from the set $\{0, 1, ..., 100\}$. You take a random ball out of the urn. It turns out to be red, you discard it. The next ball that you pick out of the black box out of the 99 remaining is:
a) Equally likely
b) More likely to be green,
c) More likely to be red.

Each of you hold one of the three conclusions: a, b and c. In order to end the argument you decide to implement a small Java program that simulates this experiment for 100k times. Write this program below.

Remember to `import java.util.Random`. Then you can do the following:
`Random r = new Random();`
`int x = r.nextInt(55);`

Picks a number uniformly at random in the set $\{0, 1, 2, ..., 54\}$.

Write your answer below:

[21]:
```
import java.util.Random;
```

```java
public static void probabilityPuzzle()
{
    Random r = new Random();
    int totalRedBalls = r.nextInt(101);
    int totalGreenBalls = 100 - totalRedBalls;

    int redBallsRemaining = totalRedBalls;
    int greenBallsRemaining = totalGreenBalls;

    int redBallsChosen = 0;
    int greenBallsChosen = 0;

    int iterations = 100000;

    for (int i = 0; i < iterations; i++)
    {
        boolean isRed = r.nextBoolean();

        if (isRed && redBallsRemaining > 0)
        {
            redBallsRemaining--;
            redBallsChosen++;
        }
        else if (!isRed && greenBallsRemaining > 0)
        {
            greenBallsRemaining--;
            greenBallsChosen++;
        }
    }

    // Output the results
    System.out.println("After " + iterations + " iterations:");
    System.out.println("Red balls picked: " + redBallsChosen);
    System.out.println("Green balls picked: " + greenBallsChosen);

    if (redBallsChosen == greenBallsChosen)
    {
        System.out.println("Both red and green balls are equally likely to be␣
↪picked next.");
    }
    else if (redBallsChosen < greenBallsChosen)
    {
        System.out.println("Green balls are more likely to be picked next.");
    }
    else
    {
        System.out.println("Red balls are more likely to be picked next.");
```

```
    }
}
```

Run your program:

[22]:
```
probabilityPuzzle();
```

```
After 100000 iterations:
Red balls picked: 9
Green balls picked: 91
Green balls are more likely to be picked next.
```

[ ]:
```

```