Lab4

March 20, 2025

1 Lab 4

Deadline: **Week 6** in your respective lab session

1.0.1 Name: Kiril Radev

1.0.2 Student ID: 241005831

You should only use things we learned up to this week (week 4), i.e. ArrayLists, HashMaps, Hashtables, etc., are not allowed. If you are unfamiliar with them, you are not expected to at this stage. Some of them will be introduced in future weeks.

1.1 Question 1 [1 mark]

Write class `Shape`.

Class `Shape` has two subclasses, `Square` and `Rectangle`.

Override the `toString()` method in class `Shape` and its subclasses. It should now return the name of the class e.g. `"Shape"`.

Write a method `populateArray`, which takes an integer as an argument specifying the size of the array, creates a new array of specified size, populates it randomly with instances of `Square` and `Rectangle` and returns it. Both classes should have an equal chance of being instantiated.

Lastly write some code to test the method you defined.

Write your answer below:

#1 ShapeOperations

[6]:
```
import java.util.Random;

public class ShapeOperations
{
        public static void main(String[] args)
        {
                Shape[] shapes = populateArray(10);

                for(Shape shape : shapes)
                {
                        System.out.println(shape.toString());
                }

                System.out.println("--------------");

                for(Shape shape : shapes)
                {
                        System.out.println(shape);
                }

                //toString() returns a String representation of an object,
                //however, since we override the toString() method does not␣
↪return the 'base' form example: "Square@2c13da15"
                //which translates to --> "object`s class name" + "@(at)" +␣
↪"hexadecimal representation of the value"
                //but now returns whatever has been altered in the method␣
↪override such as "Square", "Rectangle", "Shape"

        //--Why do both shape and shape.toSTring() return the same thing?
                //Additionally, since what we are printing is a String␣
↪representation of the value of an object
                //the System.out.println() method automatically call the␣
↪toString() method of the object to formulate it

        }//END main

        public static Shape[] populateArray(int size)
        {
                Shape[] array_to_populate = new Shape[size];
                Random random = new Random();

                for(int i = 0; i < size; i++)
                {
                        //another way is using nextBoolean to apply the equal␣
↪percent chance
                        //since it has two possible outcomes true & false
                        if(random.nextInt(2) == 0)
                        {
                                array_to_populate[i] = new Square();
                        }
                        else
                        {
                                array_to_populate[i] = new Rectangle();
                        }
                }

                return array_to_populate;
        }//END populateArray
}
```

#2 Shape

[3]:
```
public class Shape {
        //override
        public String toString() {
                return "Shape";
        }
}
```

#3 Square

[4]:
```
public class Square extends Shape {
        //override
        public String toString() {
                return "Square";
        }
}
```

#4 Rectangle

[5]:
```
public class Rectangle extends Shape {
        //override
        public String toString() {
                return "Rectangle";
        }
}
```

Run your program:

[7]:
```
ShapeOperations.main(null);
```

```
Rectangle
Square
Rectangle
Square
Rectangle
Rectangle
Rectangle
Square
Square
Rectangle
--------------
Rectangle
```

```
Square
Rectangle
Square
Rectangle
Rectangle
Rectangle
Square
Square
Rectangle
```

1.2 Question 2 [1 mark]

Write a method `separateArray` which separates an array of type `Animal` into two arrays and returns both inside another array (2D array). The first array should be of type `Dog` and contain just instances of class `Dog`, and the second should be of type `Cat` with just instances of class `Cat`.

Remember to test your code!

Write your answer below:

#1 Tester with main and separateArray

[12]:
```java
import java.util.Random;

public class Tester
{
        public static void main(String[] args)
        {
                Animal[][] animal_matrix = separateArray();

                for(Animal[] animals : animal_matrix)
                {
                        for(Animal animal: animals)
                        {
                                System.out.print(animal + " ");
                        }
                        System.out.println("");
                }
        }//END main

        public static Animal[][] separateArray()
        {
                int size = 10;
                Animal[] animals = new Animal[size];
                Random random = new Random();
                //helpful when creating the new separate finite arrays of cats␣
 ↪and dogs
```

```
                //since it is easier to have the numbers beforehand rather than␣
↪counting, then store the elements
                int number_of_dogs = 0;
                int number_of_cats = 0;

                //method assigning random object for easier input
                for(int i = 0; i < size; i++)
                {
                        if(random.nextBoolean())
                        {
                                animals[i] = new Dog();
                                number_of_dogs++;
                        }
                        else
                        {
                                animals[i] = new Cat();
                                number_of_cats++;
                        }
                }

                //-----THE ACTUAL METHOD-----

                //1. split the array
                Dog[] dogs = new Dog[number_of_dogs]; //split array type Dog
                int dog_index = 0;

                Cat[] cats = new Cat[number_of_cats]; //split array type Cat
                int cat_index = 0;

                for(int i = 0; i < size; i++)
                {
                        if(animals[i] instanceof Dog)
                        {
                                dogs[dog_index] = (Dog) animals[i];
                                dog_index++;
                        }
                        else
                        {
                                cats[cat_index] = (Cat) animals[i];
                                cat_index++;
                        }
                }

                //2. save in a matrix
                int cols = (number_of_dogs >= number_of_cats) ? number_of_dogs :
↪ number_of_cats; //() if statement ? then outcome : else outcome
                Animal[][] animals_matrix = new Animal[2][cols];
                for(int d = 0; d < number_of_dogs; d++)
                {
                        animals_matrix[0][d] = dogs[d];
                }
                for(int c = 0; c < number_of_cats; c++)
                {
                        animals_matrix[1][c] = cats[c];
                }

                return animals_matrix;
        }//END separateArray
}
```

#2 Animal

[9]:
```
public class Animal {}
```

#3 Dog

[10]:
```
public class Dog extends Animal {}
```

#4 Cat

[11]:
```
public class Cat extends Animal {}
```

Run your program:

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

```
REPL.$JShell$21$Dog@4319e004 REPL.$JShell$21$Dog@4dde0617
REPL.$JShell$21$Dog@49e18b74 REPL.$JShell$21$Dog@ce30c2c
REPL.$JShell$21$Dog@f6d474 REPL.$JShell$21$Dog@3da584c3
REPL.$JShell$19$Cat@785a0228 REPL.$JShell$19$Cat@6cccf363
REPL.$JShell$19$Cat@5a2fcffa REPL.$JShell$19$Cat@4b3ddbe0 null null
```

1.3 Question 3 [1 mark]

Write a method `countAnimalInstances` which takes an array of type `Animal` and returns an array with counts for each unique class.

For example `{new Cat(), new Dog(), new Cat()}` should return `{2, 1}`. This is because the instance of class `Cat` occurred `2` times, and the instance of class `Dog` occurred `1` time. The order of the counts should be based on the order in which instances occurred first in the array. Here, the cat occurred first; thus, it is placed at index `0`, whereas the dog occurred second; thus, it is placed at index `1`.

Another example: {new Cat(), new Pig(), new Cat(), new Cat(), new Dog(), new Cow(), new Dog()} should return {3, 1, 2, 1}.

You should assume the method must work for any number of subclasses of Animal, i.e. if we create subclasses Pig and Cow, the method should still work.

Write your answer below:

[16]:
```java
public class Animal {}
public class Cat extends Animal {}
public class Pig extends Animal {}
public class Dog extends Animal {}
public class Cow extends Animal {}
//END Animal creation

public class Tester3
{
        public static void main(String[] args)
        {
                Animal[] animals = new Animal[] {new Cat(), new Pig(), new␣
 ↪Cat(), new Cat(), new Dog(), new Cow(), new Dog()};
                int[] animals_number = countAnimalInstances(animals);

                for(int num : animals_number)
                {
                        System.out.print(num + " ");
                }
        }//END main

        public static int[] countAnimalInstances(Animal[] animals)
        {
                int size = animals.length;                              //
 ↪temporary size
                boolean[] array_of_checked_animals = new boolean[size]; //array␣
 ↪to which we map the objects
                int[] number_of_animals_by_type = new int[size];        //the␣
 ↪return array with temporary size which will be altered later
                int number_of_animal_types = 0;                         //
 ↪counts the types to later change the temporary size of the return array

                //linear search with divide and conquer technique
                for(int pivot = 0; pivot < size; pivot++)
                {
                        if(!array_of_checked_animals[pivot]) //previously␣
 ↪checked types get skipped
                        {
                                number_of_animal_types++;
                                int animal_type_counter = 1;
                                for(int checker = pivot + 1; checker < size;␣
 ↪checker++) //checks the consecutive after pivot
                                {
                    //another way is by using instanceof by checking with␣
 ↪another array that has the names if the classes stored
                                        if(!array_of_checked_animals[checker]␣
 ↪&& animals[checker].getClass().isInstance(animals[pivot]))
                                        {
                                                array_of_checked_animals[checker]␣
 ↪= true;
                                                animal_type_counter++;
                                        }
                                }
                                number_of_animals_by_type[number_of_animal_types␣
 ↪- 1] = animal_type_counter;
                        }
                }

                //remove any 0 values by creating a new array to store only the␣
 ↪counted classes
                int[] final_number_of_animals_by_type = new␣
 ↪int[number_of_animal_types];
                for(int i = 0; i < number_of_animal_types; i++)
                {
                        final_number_of_animals_by_type[i] =␣
 ↪number_of_animals_by_type[i];
                }

                return final_number_of_animals_by_type;
        }//END countAnimalInstances
}
```

Run your program:

[15]:
```
Tester3.main(null);
```

```
3 1 2 1
```

1.4 Question 4 [1 mark]

Write class `Person`, which has a private instance variable `name`, a constructor to initialise the `name` and a method `printInfo`, which prints out the person's name.

Write another class, `Student`, a subclass of a `Person`. `Student` has instance variable `SID`, constructor to initialise the `SID` and overrides method `printInfo`. It should print out the `name` and

SID.

You are expected to use the keyword `super` in your answer.

Test your code!

Write your answer below:

#1 Tester4

[21]:
```
public class Tester4
{
        public static void main(String[] args)
        {
                Person person = new Person("Kalin");
                person.printInfo();

                Student student = new Student("Irem","241002910");
                student.printInfo();
        }//END main
}//END Tester4
```

#2 Person

[19]:
```
public class Person
{
        private String name;

        //constructor
        Person (String new_name)
        {
                this.setName(new_name);
        }
        //encapsulation
        //access.or method
        public String getName() {
                return name;
        }
        //mutator method
        public void setName(String name) {
                this.name = name;
        }

        //method to print the name
        public void printInfo()
        {
                System.out.println("Name: " + getName());
        }//END printInfo
}//END Person
```

#3 Student

[20]:
```
public class Student extends Person
{
        String SID;

        //constructor
        Student (String new_name, String new_SID)
        {
                super(new_name);
                this.SID = new_SID;
        }

        //override
        public void printInfo()
        {
                System.out.println("Name: " + getName());
                System.out.println(" SID: " + SID);
        }//END printInfo
}//END Student
```

Run your program:

[22]:
```
Tester4.main(null);
```

```
Name: Kalin
Name: Irem
 SID: 241002910
```

1.5 Question 5 [1 mark]

A unit fraction contains 1 in the numerator. The decimal representation of the unit fractions with denominators 2 to 10 are given:

$1/2 = 0.5$
$1/3 = 0.(3)$
$1/4 = 0.25$
$1/5 = 0.2$
$1/6 = 0.1(6)$
$1/7 = 0.(142857)$
$1/8 = 0.125$
$1/9 = 0.(1)$
$1/10 = 0.1$

Where 0.1(6) means 0.166666..., and has a 1-digit recurring cycle. It can be seen that 1/7 has a 6-digit recurring cycle.

Find the value of $d < 1000$ for which $1/d$ contains the longest recurring cycle in its decimal fraction part.

To understand how one can compute the length of the recurring cycle of $1/d$ we have to look at the division algorithm in more detail. Let us take the case of $1/7$. The first part of the algorithm multiplies the numerator by 10 until it becomes larger than the denominator. In our case we multiply $1 \cdot 10$ (if we had $d > 10$ we would multiply by 100, and if we had $d > 100$ we would multiply by 1000).

Now the division algorithm proceeds as follows - First $10/7$ is 1 with a remainder of 3, - Then $10 \cdot 3/7$ is 4 with a remainder of 2, - Then $10 \cdot 2/7$ is 2 with a remainder of 6, - Then $10 \cdot 6/7$ is 8 with a remainder of 4, - Then $10 \cdot 4/7$ is 5 with a remainder of 5, - Then $10 \cdot 5/7$ is 7 with a remainder of 1, - And $10 \cdot 1/7$ is 1 with a remainder of 3.

Now it is clear that this process would repeat, and that the length of this cycle is 6. The important observation here is that if r_1 is the first remainder, in the next step we are doing $r_2 = (10 \cdot r_1)\%7$ to get the second remainder, and so on, $r_n = (10 \cdot r_{n-1})\%7$. The sequence $r_1, r_2, \ldots$ has to have a cycle because its entries are numbers between 0 and 6 (because we are taking reminders mod 7).

Write your answer below:

[1]:
```java
public class Tester5
{
        public static void main(String[] args) throws InterruptedException
        {
                int size = 1000;
                int number_with_longest_reccuring_cycle = 0;
                String cuttenr_longest_recuring_cycle = "";
                //cuttenr_longest_recuring_cycle = calculateReccuringCycle(4);

                for(int i = 2; i < size; i++)     //exclude 1 since it does not␣
 ↪change the number
                {
                        if(i % 2 == 0 || i % 5 == 0) //multiples and powers of␣
 ↪2 & 5 result into short or terminating cycles
                        {
                                continue;
                        }

                        String temporary = calculateReccuringCycle(i);
                        if(cuttenr_longest_recuring_cycle.length() < temporary.
 ↪length())
                        {
                                cuttenr_longest_recuring_cycle = temporary;
                                number_with_longest_reccuring_cycle = i;
                        }
                }

                //Number Theory: the length of the longest recurring cycle is␣
 ↪expected to be at max n-1
```

```
                System.out.println(number_with_longest_reccuring_cycle + " has␣
↪the longest recurring cycle of length: " + cuttenr_longest_recuring_cycle.
↪length());
        }//END main

        public static String calculateReccuringCycle(int divisor)
        {
                int divident = 1;
                int first_remainder = 0;
                int remainder = 0;
                int quotient = 0;
                String repeated_decimal = "";
                int m = 0;

                if(divisor < 10)
                {
                        m = 10;
                }
                else if(divisor < 100)
                {
                        m = 100;
                }
                else
                {
                        m = 1000;
                }

                //first decimal
                quotient = (m * divident) / divisor;
                first_remainder = (m * divident) % divisor;
                repeated_decimal += quotient;

                //second decimal
                quotient  = (10 * first_remainder) / divisor;
                remainder = (10 * first_remainder) % divisor;
                repeated_decimal += quotient;

                while(remainder != first_remainder)
                {
                        quotient  = (10 * remainder) / divisor;
                        remainder = (10 * remainder) % divisor;
                        //to omit the writing of the first_decimal
                        if(remainder != first_remainder)
                        {
                                repeated_decimal += quotient;
                        }
                }
                return repeated_decimal;
        }//END calculateReccuringCycle
}
```

Run your program:

[2]:
```
Tester5.main(null);
```

```
983 has the longest recurring cycle of length: 982
```