Lab5

March 6, 2024

1 Lab 5

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

1.0.1 Name:Hansel Rodrigues

1.0.2 Student ID: 230603866

You should only use things we learned up to this week (week 5), 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 an interface `SquareDrawer` that contains two methods with return types `void`, `drawSquare` and `drawSpecialSquare`.

Write an interface `StairCaseDrawer` which contains one method with return type `void drawStairCase`.

Write a class `Drawer` with three class variables `sideLength`, `filler`, and `special` of type int, char, and char, respectively. This class should also contain a constructor which takes three parameters and sets fields to these values.

Test your code!

Write your answer below:

[17]:
```
interface SquareDrawer
{
   public void drawSquare();
   public void drawSpecialSquare();
}
```

[18]:
```
interface StairCaseDrawer
{
   public void drawStairCase();
}
```

[33]:
```
class Drawer implements SquareDrawer, StairCaseDrawer
{
    int sideLength;
    char filler;
    char special;
    public Drawer(int sideLength, char filler, char special)
    {
        this.sideLength = sideLength;
        this.filler = filler;
        this.special = special;
    }
    @Override
    public void drawSquare()
    {
        for (int i = 0; i < sideLength; i++)
        {
            for (int j = 0; j < sideLength; j++)
            {
                System.out.print(filler);
            }
            System.out.println();
        }
    }

   @Override
   public void drawSpecialSquare()
   {
       for (int i = 0; i < sideLength; i++)
        {
            for (int j = 0; j < sideLength; j++)
            {
                System.out.print(special);
            }
            System.out.println();
        }

   }
   @Override
   public void drawStairCase()
   {
       for (int i = 0; i < sideLength; i++)
       {
            for (int j = 0; j <= i; j++)
            {
                System.out.print(filler + " ");
            }
            System.out.println();
```

```
        }
    }
}
class Testing
{
    public static void main(String[] args)
    {
        Drawer drawer = new Drawer(5, '*', '$');

        System.out.println("Square:");
        drawer.drawSquare();

        System.out.println("\nSpecial Square:");
        drawer.drawSpecialSquare();

        System.out.println("\nStaircase:");
        drawer.drawStairCase();
    }
}
```

Run your program:

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

```
Square:
*****
*****
*****
*****
*****

Special Square:
$$$$$
$$$$$
$$$$$
$$$$$
$$$$$

Staircase:
*
* *
* * *
* * * *
* * * * *
```

1.2 Question 2 [1 mark]

Write a class `ShapeDrawer1`, a subclass of a Drawer, and implement the interface `SquareDrawer`.

This class should make use of its superclass's constructor.

`drawSquare` should print a square out of character stored inside `filler` with a side specified by `sideLength`.

For example `drawSquare` where `sideLength = 5` and `filler = '#'` should print out the following:

```
#####
#####
#####
#####
#####
```

`drawSpecialSquare` should print a square like `drawSquare`, but now diagonals should be made out of characters stored inside `special`.

For example `drawSpecialSquare` where `sideLength = 5`, `filler = '#'` and `special = 'X'` should print out the following:

```
X###X
#X#X#
##X##
#X#X#
X###X
```

You can assume the `sideLength` is always odd.

Remember to test your code!

Write your answer below:

[133]:
```
class ShapeDrawer1 extends Drawer implements SquareDrawer
{
    public ShapeDrawer1(int sideLength, char filler, char special)
    {
        super(sideLength, filler, special);
    }
    @Override
    public void drawSquare()
    {
        for (int i = 0; i < sideLength; i++)
        {
            for (int j = 0; j < sideLength; j++)
            {
                System.out.print(filler);
            }
            System.out.println();
        }
    }
```

```
    @Override
    public void drawSpecialSquare()
    {
        for (int i = 0; i < sideLength; i++)
        {
            for (int j = 0; j < sideLength; j++)
            {
                if (i == j || i == sideLength - 1 - j)
                {
                    System.out.print(special);
                }
                else
                {
                    System.out.print(filler);
                }
            }
            System.out.println();
        }
    }
}
class Test
{
    public static void main(String[] args)
    {
        ShapeDrawer1 drawer = new ShapeDrawer1(5, '#', 'X');

        System.out.println("Drawing Square:");
        drawer.drawSquare();
        System.out.println();

        System.out.println("Drawing Special Square:");
        drawer.drawSpecialSquare();
        System.out.println();
    }
}
```

Run your program:

[134]:
```
Test.main(null);
```

```
Drawing Square:
#####
#####
#####
#####
#####

Drawing Special Square:
```

```
X###X
#X#X#
##X##
#X#X#
X###X
```

1.3 Question 3 [1 mark]

Copy the class from Question 2 and rename `ShapeDrawer1` to `ShapeDrawer2` where appropriate.

`ShapeDrawer2` should implement both `SquareDrawer` and `StaircaseDrawer`.

The `StairCase` drawer should print out the staircase out of `filler` with each step of size `sideLength` in both dimensions with a number of steps specified by `sideLength`.

For example where `sideLength = 5` and `filler = #` should print out the following:

```
#####
#####
#####
#####
#####
##########
##########
##########
##########
##########
###############
###############
###############
###############
###############
####################
####################
####################
####################
####################
#########################
#########################
#########################
#########################
#########################
```

Test whether all methods in `ShapeDrawer2` work as expected!

Write your answer below:

[203]:
```
class ShapeDrawer2 extends Drawer implements SquareDrawer, StairCaseDrawer
{
    public ShapeDrawer2(int sideLength, char filler, char special)
    {
        super(sideLength, filler, special);
    }
    @Override
    public void drawStairCase()
    {
        for (int set = 1; set <= sideLength; set++)
        {
            for (int row = 1; row <= sideLength; row++)
            {
                for (int col = 1; col <= sideLength * set; col++)
                {
                    System.out.print(filler);
                }
                System.out.println();
            }
        }
    }
}
public class Test3 {
    public static void main(String[] args) {
        ShapeDrawer2 shapeDrawer = new ShapeDrawer2(5, '#', 'X');

        shapeDrawer.drawStairCase();
    }
}
```

Run your program:

[204]:
```
Test3.main(null);
```

```
#####
#####
#####
#####
#####
##########
##########
##########
##########
##########
###############
###############
###############
###############
```

```
###############
###################
###################
###################
###################
###################
########################
########################
########################
########################
########################
```

1.4 Question 4 [1 mark]

You are given three methods below, one for each `String`, `Integer`, and `Boolean`, respectively, that takes two objects and checks whether they store the same value.

Write one method that does the same check but works for all types.

You are expected to test your code thoroughly!

Hint: Use generics.

Write your answer below:

[1]:
```java
public static boolean areEqual(String s1, String s2) {
    return s1.equals(s2);
}

public static boolean areEqual(Integer i1, Integer i2) {
    return i1.equals(i2);
}

public static boolean areEqual(Boolean b1, Boolean b2) {
    return b1.equals(b2);
}
```

[208]:
```java
public class CheckingEquality
{
    public static <T> boolean areEqual(T obj1, T obj2)
    {
        if (obj1 == null && obj2 == null)
        {
            return true;
        }
        if (obj1 == null || obj2 == null)
        {
            return false;
```

```
        }
        return obj1.equals(obj2);
    }
    public static void main(String[] args)
    {
    System.out.println(areEqual("hello", "hello"));
    System.out.println(areEqual("hello", "world"));
    System.out.println(areEqual(5, 5));
    System.out.println(areEqual(5, 4));
    System.out.println(areEqual(null, null));
    System.out.println(areEqual(true, false));
    }
}
```

Run your program:

[209]:
```
CheckingEquality.main(null);
```

```
true
false
true
false
true
false
```

1.5 Question 5 [1 mark]

Write a method `isReachable` which takes two `Node` arguments, one for the start node and one for the end node. The method returns `true` if and only if it is possible to go from the node `start` to the node `end` in the directed graph, following the arrows.

You are provided with a class `Node`, which represents a node within a graph, `label` corresponds to the "name" of the node, and the array `outgoing` is all the nodes that are connected to the current node by outgoing arrows.

The image below shows an example directed graph that we also coded below to make testing easier.

We wrote some tests for you, but you are expected to write a few more.

Write your answer below:

[326]:
```
class Node {
    String label;
    Node[] outgoing;

    Node(String label) {
        this.label = label;
        this.outgoing = null;
```

```
    }
    public void linkTo(Node n) {
        if(outgoing == null) {
            outgoing = new Node[1];
            outgoing[0] = n;
            return;
        }

        Node[] newOutgoing = new Node[outgoing.length+1];
        for(int i = 0; i < outgoing.length; i++) {
            newOutgoing[i] = outgoing[i];
        }
        newOutgoing[outgoing.length] = n;
        outgoing = newOutgoing;
    }
}
```

[349]:
```
public static boolean isReachable(Node start, Node end)
{
    if (start == null)
    {
        return false;
    }
    return nodePath(start, end, new boolean[100], new boolean[100]);
}
public static boolean nodePath(Node current, Node end, boolean[] visited,␣
 ↪boolean[] checked)
{
    if (current == end)
            return true;
        int currentIndex = getIndex(current.label);
        if (checked[currentIndex])
            return false;
        visited[currentIndex] = true;
        if (current.outgoing != null)
        {
            for (Node neighbor : current.outgoing)
            {
                int neighborIndex = getIndex(neighbor.label);
                if (!visited[neighborIndex] && nodePath(neighbor, end, visited,␣
 ↪checked))
                {
                    return true;
                }
            }
        }
        visited[currentIndex] = false;
```

```
        checked[currentIndex] = true;
        return false;
    }
    private static int getIndex(String label)
        {

        Node[] allNodes = getAllNodes();
        for (int i = 0; i < allNodes.length; i++)
        {
            if (allNodes[i].label.equals(label))
            {
                return i;
            }
        }
        return -1;
    }

    private static Node[] getAllNodes()
    {

        Node[] nodes = {
            new Node("A"),
            new Node("B"),
            new Node("C"),
            new Node("D"),
            new Node("E"),
            new Node("F")
        };
        return nodes;
    }
```

[352]:
```
Node a = new Node("A");
Node b = new Node("B");
Node c = new Node("C");
Node d = new Node("D");
Node e = new Node("E");
Node f = new Node("F");

a.linkTo(b);
b.linkTo(c);
c.linkTo(e);
e.linkTo(f);
e.linkTo(d);
d.linkTo(b);

System.out.println(isReachable(a, e)); // true
System.out.println(isReachable(f, a)); // false
```

```
System.out.println(isReachable(a, f));
System.out.println(isReachable(f,b));

// MORE TESTS HERE
```

```
true
false
true
false
```

[ ]: