Lab 2

PROGRAMMING III – IT7374

Rodrigo Costa #2141030

2016

Contents

[Exercise 2 3](#_Toc445923056)

[Read chapter 10 (only p.449 – 454) from Bruce Eckel Implement abstract class Shape. Derive Square and Circle from Shape. Create ArrayList of Shapes and populate it with Square and Circle objects. Write Code to count the number of each type of shape in the array list. 3](#_Toc445923057)

[Exercise 4 4](#_Toc445923058)

[Taking a Java Class and finding any information with the implementation of a reflection. 4](#_Toc445923059)

# Exercise 2

## Read chapter 10 (only p.449 – 454) from Bruce Eckel Implement abstract class Shape. Derive Square and Circle from Shape. Create ArrayList of Shapes and populate it with Square and Circle objects. Write Code to count the number of each type of shape in the array list.

class Shape {

void draw() { System.out.println(this + ".draw()"); }

}

class Circle extends Shape {

public String toString() { return "Circle"; }

}

class Square extends Shape {

public String toString() { return "Square"; }

}

import java.util.\*;

public class ShapesList {

public static void main(String[] args) {

ArrayList<Shape> arraylist = new ArrayList<Shape>();

arraylist.add(new Shape());

arraylist.add(new Shape());

arraylist.add(new Shape());

arraylist.add(new Shape());

arraylist.add(new Shape());

arraylist.add(new Shape());

Object[] shapeList = {

new Circle(),

new Square()

};

for(int i = 0; i < shapeList.length; i++){

((Shape)shapeList[i]).draw();

{

"Circle.draw()",

"Square.draw()",

"Triangle.draw()"

});

};

}};

//Wouldn't be better to implement Shape as an interface? Since there is a lack of methods, the implementation of the Shape as an interface would allow it to be easier to inherit from other places if needed without differences in its majority.

# Exercise 4

## Taking a Java Class and finding any information with the implementation of a reflection.

import java.io.\*;

import java.lang.reflect.\*;

public class reflectExample

{

public reflectExample()

{

Class className.getName();

}

public static void main(String[] args)

{

File f = new File("TheFile.txt"); // text file in current project directory

String[] vals; // tokens from each line of the file

Class currentClass = null;

Object newObj = null;

Method reqMethod = null;

try

{ // throws FileNotFoundException

BufferedReader input = new BufferedReader(new FileReader(f));

String str = new String();

while ( (str = input.readLine() ) != null) // throws IOException

{

vals = str.split(" "); // split line into tokens

if (vals.length == 1) // a class name - not a set of attributes

// use the name of the class to find the Class object

currentClass = Class.forName(vals[0]);

else // throws classNotFoundException

{ // more than one token indicates attributes - not a class name

// so create an object –method is user-defined below

newObj = createObject(currentClass, vals);

if (newObj != null) // throws NoSuchMethodException

{ reqMethod = currentClass.getMethod("Print", null);

// call the found method i.e. Print on - newObject.print();

reqMethod.invoke(newObj, null); // throws IllegalAccessException

}

}

}

}

catch (ClassNotFoundException e)

{ System.out.println ("ClassNotFoundException"); }

catch (NoSuchMethodException nsme)

{ System.out.println ("NoSuchMethodException"); }

catch (InvocationTargetException jte)

{ System.out.println ("InvocationTargetException"); }

catch (IllegalAccessException iaex)

{ System.out.println ("llegalAccessException"); }

catch (FileNotFoundException fnfe)

{ System.out.println ("FileNotFoundException"); }

catch (IOException ioe)

{ System.out.println ("IOException"); }

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public static Object createObject(Class currentClass, String [] strArgs)

{

// static because it is called from a static context i.e. main()

Class[] paramTypes = null;

Object[] objArgs = null;

// Find all constructors for currentClass

Constructor[] cons = currentClass.getConstructors();

for (int i = 0; i < cons.length; i++) // Iterate through all constructors

{

paramTypes = cons[i].getParameterTypes(); // 'Class' for each param type

objArgs = new Object[paramTypes.length]; // array of Obj for params

// find a constructor having the same number of args as read from file

if (paramTypes.length == strArgs.length)

{

for (int j = 0; j < paramTypes.length; j++) // wrap all primitives

{ if (paramTypes[j] == String.class)

objArgs[j] = strArgs[j];

else if (paramTypes[j].getName().equals("float"))

objArgs[j] = new Float(Float.parseFloat(strArgs[j]));

else if (paramTypes[j].getName().equals("int"))

objArgs[j] = new Integer(Integer.parseInt(strArgs[j]));

else if (paramTypes[j].getName().equals("boolean"))

objArgs[j] = new Boolean(Boolean.getBoolean(strArgs[j]));

else if (paramTypes[j].getName().equals("char"))

objArgs[j] = new Character(strArgs[j].charAt(0));

}

try { // construct a new instance passing args as objects

return cons[i].newInstance(objArgs);

}

catch (InstantiationException ie)

{ }

catch (IllegalAccessException ie)

{ }

catch (InvocationTargetException ie)

{ }

}

}

return null;

}

}