**Programming languages / Java (BSc, 18) 6th lab**

**Instructions**

Following the propagation: fixing compile time error

import [java.io](http://java.io/). IOException;

class TestTime {

public Time readTime( String fname ) throws IOException {

... new java.io.FileReader(fname) ...

}

public static void main( String[] args ) throws IOException {

TestTime tt = new TestTime();

Time wakeUp = tt.readTime("wakeup.txt");

wakeUp.aMinutePassed();

}

}

Exception handling

import [java.io](http://java.io/). IOException;

class TestTime {

public Time readTime( String fname ) throws IOException {

... new java.io.FileReader(fname) ...

}

public static void main( String[] args ){

TestTime tt = new TestTime();

try {

Time wakeUp = tt.readTime("wakeup.txt");

wakeUp.aMinutePassed();

} catch( IOException e ){

System.err.println("Could not read wake-up time.");

}

}

}

Exception handling

The program continues running despite the error

public class Receptionist {

...

public Time[] readWakeupTimes( String[] fnames ){

Time[] times = new Time[fnames.length];

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

try {

times[i] = readTime(fnames[i]);

} catch( java.io.IOException e ){

times[i] = null; // no-op

System.err.println("Could not read " + fnames[i]);

}

}

return times; // maybe sort times before returning?

}

}

<https://stackoverflow.com/questions/11589302/why-is-throws-exception-necessary-when-calling-a-function>  
<https://stackoverflow.com/questions/2683958/why-is-nullpointerexception-not-declared-as-a-checked-exception>  
<https://stackoverflow.com/questions/297303/printwriter-and-printstream-never-throw-ioexceptions>

**Task 1**

**a**

Modify **Task 5** of the **3ed lab** so that any Point object may be constructed without parameters! In this case, fields x and y shall be initialised to 0. Provide setter methods for the x and y fields of the Point class! Modify the Circle class's constructor so that it copies the given Point object instead of storing a reference to it!

Circles shall have labels! Extend the Circle class with a field capable of storing a label ( String )! Adjust the constructor, so that the label may be specified during construction. Make sure to update the toString() method's output with the label! Create a static final field for the class called defaultLabel with the value "unnamed" . When the label provided during construction is null the label of the constructed Circle object shall be the defaultLabel .

Provide the Circle class with a constructor that receives two double parameters ( x , and y ) instead of a Point and delegates construction to the constructor we created earlier!

Create a static method in the Circle class called readFromFile() which loads the data of a circle from a file (provided as parameter). The file should contain the coordinates *x* and *y*, the *radius*, and the *label*, all separated by newlines. The method shall construct a Circle object using these data and return its reference. Any exception that can arise during these operations should be handled by the caller.

Create a method called saveToFile() in the Circle class that saves the current Circle object to the file (provided as parameter). Exceptions during file operations should not be handled in the method. However, exceptions during write operation must not result in the loss of already written data, therefore make sure the PrintWriter object is properly closed!

package circle;

import java.io.File;

import java.io.FileReader;

import java.io.BufferedReader;

import java.io.PrintWriter;

import java.io.FileNotFoundException;

import java.io.IOException;

import circle.utils.Point;

public class Circle {

private Point center;

private double radius;

private String label;

private static final String defaultLabel = "unnamed";

public Circle(Point center, double radius, String label) {

this.center = new Point(center);

this.radius = radius;

if (label == null)

this.label = defaultLabel;

else

this.label = label;

}

public Circle(double x, double y, double radius, String label) {

this(new Point(x, y), radius, label);

}

public static Circle readFromFile(String filename) throws FileNotFoundException, IOException {

File input = new File(filename);

BufferedReader bf = null;

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

//System.err.println("Successfully opened file: " + filename);

double x = Double.parseDouble(bf.readLine());

double y = Double.parseDouble(bf.readLine());

double radius = Double.parseDouble(bf.readLine());

String label = bf.readLine();

bf.close();

return new Circle(x, y, radius, label);

}

/\*

// not sure h close

public void saveToFile(String filename) throws FileNotFoundException {

File output = new File(filename);

PrintWriter pw = new PrintWriter(output);

pw.println(center.getX());

pw.println(center.getY());

pw.println(radius);

pw.println(label);

pw.close();

}

\*/

public void saveToFile(String filename) throws FileNotFoundException {

File output = new File(filename);

try (PrintWriter pw = new PrintWriter(output)) {

pw.println(center.getX());

pw.println(center.getY());

pw.println(radius);

pw.println(label);

}

}

public void enlarge(double f) {

radius \*= f;

}

public double getArea() {

return Math.PI \* radius \* radius;

}

public double getRadius() {

return radius;

}

public String toString() {

return label + " : " + "(" + center.getX() + "," + center.getY() + "), r = " + radius;

}

}

package circle;

import java.io.FileNotFoundException;

import java.io.IOException;

import circle.utils.Point;

public class Main {

public static void main(String[] args) {

Point center = new Point();

Circle c1 = new Circle(center, 10, null);

System.out.println(c1);

c1.enlarge(3);

//center.setX(5); // Circle konstruktorĂĄban this.center = new Point(center); vs this.center = center;

System.out.println(c1);

c1.enlarge(2);

System.out.println("circle's radius is " + c1.getRadius());

Circle c2 = null;

try {

//c1.saveToFile("");

c1.saveToFile("copy1.txt");

c1.saveToFile("copy2.txt");

c1.saveToFile("copy3.txt");

c2 = Circle.readFromFile("in.txt");

}

catch (FileNotFoundException e) {

System.err.println("Error: Unable to access file, " + e.getMessage());

}

catch (IOException e) {

System.err.println("Error: IO error occured, " + e.getMessage());

}

System.out.println(c2);

}

}

**package circle.utils;**

**public class Point {**

**private double x, y;**

**//public Point () {}**

**public Point() {**

**this.x = this.y = 0;**

**}**

**public Point(double x, double y) {**

**this.x = x;**

**this.y = y;**

**}**

**public Point(Point that) {**

**this.x = that.x;**

**this.y = that.y;**

**}**

**public double getX() {**

**return x;**

**}**

**public double getY() {**

**return y;**

**}**

**public void setX(double x) {**

**this.x = x;**

**}**

**public void setY(double y) {**

**this.y = y;**

**}**

**public void move(double dx, double dy) {**

**x += dx;**

**y += dy;**

**}**

**public void mirror(double cx, double cy) {**

**x = 2 \* cx - x;**

**y = 2 \* cy - y;**

**}**

**public void mirror(Point that) {**

**x = 2 \* that.x - x;**

**y = 2 \* that.y - y;**

**}**

**public double distance(Point that) {**

**double dx = x - that.x;**

**double dy = y - that.y;**

**return Math.sqrt(dx\*dx + dy\*dy);**

**}**

**}**

Copy1.txt

0.0

0.0

60.0

Unnamed

Copy2.txt

0.0

0.0

60.0

Unnamed

Copy3.txt

0.0

0.0

60.0

Unnamed

**In.txt**

**2**

**2.3**

**10**

**circle2Label**

**b**

Create a modified version of **Task 1 (a)** in which readFromFile() tries to handle exceptional situations arising during file operations. If reading the input file fails due to an exceptional event, the method shall return an object filled with zero-like data.

package circle;

import java.io.File;

import java.io.FileReader;

import java.io.BufferedReader;

import java.io.PrintWriter;

import java.io.FileNotFoundException;

import java.io.IOException;

import circle.utils.Point;

public class Circle {

private Point center;

private double radius;

private String label;

private static final String defaultLabel = "unnamed";

public Circle(Point center, double radius, String label) {

this.center = new Point(center);

this.radius = radius;

if (label == null)

this.label = defaultLabel;

else

this.label = label;

}

public Circle(double x, double y, double radius, String label) {

this(new Point(x, y), radius, label);

}

public static Circle readFromFile(String filename) {

File input = new File(filename);

BufferedReader bf = null;

Circle result = null;

try {

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

//System.err.println("Successfully opened file: " + filename);

double x = Double.parseDouble(bf.readLine());

double y = Double.parseDouble(bf.readLine());

double radius = Double.parseDouble(bf.readLine());

String label = bf.readLine();

result = new Circle(x, y, radius, label);

bf.close();

}

catch (IOException | NumberFormatException e) {

result = new Circle(0, 0, 0, null);

}

return result;

}

public void saveToFile(String filename) throws FileNotFoundException {

File output = new File(filename);

try (PrintWriter pw = new PrintWriter(output)) {

pw.println(center.getX());

pw.println(center.getY());

pw.println(radius);

pw.println(label);

}

}

public void enlarge(double f) {

radius \*= f;

}

public double getArea() {

return Math.PI \* radius \* radius;

}

public double getRadius() {

return radius;

}

public String toString() {

return label + " : " + "(" + center.getX() + "," + center.getY() + "), r = " + radius;

}

}

package circle;

import java.io.FileNotFoundException;

import circle.utils.Point;

public class Main {

public static void main(String[] args) {

Point center = new Point();

Circle c1 = new Circle(center, 10, null);

System.out.println(c1);

c1.enlarge(3);

//center.setX(5);

// Circle konstruktorĂĄban this.center = new //

//Point(center); vs this.center = center;

System.out.println(c1);

c1.enlarge(2);

System.out.println("circle's radius is " + c1.getRadius());

try {

//c1.saveToFile("");

c1.saveToFile("copy1.txt");

c1.saveToFile("copy2.txt");

c1.saveToFile("copy3.txt");

}

catch (FileNotFoundException e) {

System.err.println("Error: Unable to write file, " + e);

}

Circle c2 = Circle.readFromFile("in.txt");

System.out.println(c2);

}

}

package circle.utils;

public class Point {

private double x, y;

//public Point () {}

public Point() {

this.x = this.y = 0;

}

public Point(double x, double y) {

this.x = x;

this.y = y;

}

public Point(Point that) {

this.x = that.x;

this.y = that.y;

}

public double getX() {

return x;

}

public double getY() {

return y;

}

public void setX(double x) {

this.x = x;

}

public void setY(double y) {

this.y = y;

}

public void move(double dx, double dy) {

x += dx;

y += dy;

}

public void mirror(double cx, double cy) {

x = 2 \* cx - x;

y = 2 \* cy - y;

}

public void mirror(Point that) {

x = 2 \* that.x - x;

y = 2 \* that.y - y;

}

public double distance(Point that) {

double dx = x - that.x;

double dy = y - that.y;

return Math.sqrt(dx\*dx + dy\*dy);

}

}

Copy1.txt

0.0

0.0

60.0

unnamed

Copy2.txt

0.0

0.0

60.0

unnamed

Copy3.txt

0.0

0.0

60.0

unnamed

in.txt

2

2.3

10

circle2Label

**Instructions**

JavaDoc format:

/\*\*

\* ...

\* ...

\*/

How should Java documentation comments written (like block tags, inline tags)?

@param, @return, @throws etc.- block tags

{@code}, {@link} - inline tags

numbers/Rational.java:

package numbers;

public class Rational {

/\*\*

- Set {@code this} to {@code this} \* {@code that}.

- @param that Non-null reference to a rational number,

- it will not be changed in the method.

- @throws NullPointerException When {@code that} is null.

\*/

public void multiplyWith( Rational that ){

this.numerator \*= that.numerator;

this.denominator \*= that.denominator;

}

}

Generate HTML: javadoc Rational.java

**Task 2**

Modify the 3rd task of the 5th lab in the following way:

Organize the logic already written into class methods. The method should also gain beside the command-line parameters a double value. It should check whether the result of the operation done rounded downwards on the two arguments is equal to this value (return a boolean value). The method should not handle exceptions, this should be done by the caller (in the main() ).

Write a javadoc to the method about its functionality! Use at least the followings:

* @param
* @returns
* @throws

Create documentation for the class as well. It should contain the @author , @version and @since tags.

Generate HTML documentation for your program with the javadoc tool!

**/\*\***

**\* @author author\_name**

**\* @since 1.0**

**\* @version 1.0**

**\*/**

**public class JavadocEx {**

**/\*\***

**\* The starting point of the Java application.**

**\* @param args commandline arguments**

**\*/**

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

**try {**

**//String[] example = {"50", "-", "51.2"};**

**//System.out.println(calculateFromArgs(example, -2)); //50 - 51.2 //true**

**System.out.println(calculateFromArgs(args, -2));**

**}**

**catch (NumberFormatException e) {**

**System.out.println("Invalid number format.");**

**System.out.println(e.getMessage());**

**}**

**catch (IllegalArgumentException e) {**

**System.out.println("Invalid program arguments provided.");**

**System.out.println(e.getMessage());**

**}**

**catch (ArithmeticException e) {**

**System.out.println("Arithmetic error occured.");**

**System.out.println(e.getMessage());**

**}**

**}**

**/\*\***

**\* <p>Performs the mathematical operation denoted by the third element of the args array on the first**

**\* and second element and returns {@code true} if the result of the calculation rounded down to its nearest**

**\* neighbouring integer is equal to the second method parameter or {@code false} otherwise.</p>**

**\* @param args array of arguments**

**\* @param result awaited result after the operation is performed**

**\* @throws IllegalArgumentException if there aren't exactly 3 arguments provided**

**\* @throws ArithmeticException if there is an attempt of dividing by zero**

**\* @return the logical boolean value of the equity check between the input and the desired result**

**\* @since 1.0**

**\*/**

**public static boolean calculateFromArgs(String[] args, double result) {**

**if (args.length != 3) {**

**throw new IllegalArgumentException("Not exactly three arguments provided.");**

**} else {**

**double a = Double.parseDouble(args[0]);**

**char m = args[1].charAt(0);**

**double b = Double.parseDouble(args[2]);**

**switch (m) {**

**case '+':**

**return result == Math.floor(a + b);**

**case '-':**

**return result == Math.floor(a - b);**

**case '\*':**

**return result == Math.floor(a \* b);**

**case '/':**

**if (b == 0) {**

**throw new ArithmeticException("Division by zero");**

**}**

**return result == Math.floor(a / b);**

**default:**

**throw new IllegalArgumentException("Unknown operation");**

**}**

**}**

**}**

**}**

**Instructions**

public enum Day {

SUNDAY, MONDAY, TUESDAY, WEDNESDAY,

THURSDAY, FRIDAY, SATURDAY

}

//...

Day d1 = Daay. MONDAY ;

Day d2 = Day.valueOf("MONDAY");

values() static method - returns every enum value as an array. ordinal() object method - returns the order of the specified value in the enum definition (starts with 0).

<https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html>

**Task 3**

Create the WildAnimal enum in the WildAnimal.java file, it should contain four values: monkey, elephant, giraffe and racoon.

These animals get two values in their constructors, which fruit do they like and the ideal portion for a day of that fruit.

Create the listAllAnimals() method, which returns a string like this:

"*animal ordinal*: *name of the animal* desires eating *name of the fruit* a week."

For example, elephants like to eat 30 portions of raspberries a day:

"2: ELEPHANT desires eating 210 raspberries a week."

The enum should be iterated using the values() , it's ordinal by the ordinal() methods.

Create a toString() method, which returns the informations stored by the specific enum value.

Try and test your enum and the methods in a Main class.

public enum WildAnimal {

MONKEY("bananas", 5),

ELEPHANT("raspberries", 30),

GIRAFFE("apples", 10),

RACCOON("walnuts", 20);

private final String food;

private final int amount;

WildAnimal(String food, int amount) {

this.food = food;

this.amount = amount;

}

public static String listAllAnimals() {

StringBuilder sb = new StringBuilder();

for (WildAnimal animal : WildAnimal.values()) {

sb.append(animal.ordinal()).append(": ");

sb.append(animal.name());

sb.append(" desires eating ").append(animal.amount \* 7).append(" ");

sb.append(animal.food).append(" a week.");

sb.append(System.lineSeparator());

}

return sb.toString();

}

public String toString() {

return "A(n) " + this.name() + " eats " + this.amount + " " + this.food + " a day.";

}

}

public class Main {

public static void main(String[] args) {

System.out.println(WildAnimal.ELEPHANT.toString());

System.out.println(WildAnimal.listAllAnimals());

}

}