Java TestEn : Analytic Company GmbH

Name: Traian GEICU

Mail:geicutraian@yahoo.com

Time to Complete : estimate around one and half day (12 hours)

Full code also available in attached jar (plain java only)

Additional explanation are available as comments on code

Environment: Win8, Eclipse, java 9 compiler

1 Regular expressions
Define a regular expression that matches the phrases “damage”, “minor damages” and “heavy damage” but not the phrase “no damages”.
Write a method with a string as argument. The output has to be the result whether the string matches or not.

Solution:

```
package testen.java;

public class RegEx
{
/* assume that also [minor damage] will fit on pattern and no extra blank spaces
s{1} : one occurrence of blank is mandatory
? : optional (0 or 1 time)
        |: either
        ()? : all pattern optional */

final static String pattern_1 = "((minor|heavy)\\s{1})?damage(s)?";
// assume that [minor damage] will not fit on pattern and no extra blank spaces (head, tail, middle)
// do not have to many nice option of adding "s" just for minor
// and here the raw expression is a good option
// sometimes simple(direct) way are better for simple things
final static String pattern_2 = "((heavy )?damage|minor damages)";

public static String getMatch(String str, String pattern)
        {
                return str.matches(pattern) ?
                                "match on #" + pattern + "# for "+str :
"no match on #" + pattern + "# for " +str;

        }

public static void main(String... args)
        {
```

```
                System.out.println(getMatch("damage", pattern_1));
                System.out.println(getMatch("no damage", pattern_1));
                System.out.println(getMatch("minor damage", pattern_1));
                System.out.println(getMatch("minor damages", pattern_1));
                System.out.println(getMatch("heavy damage", pattern_1));
                System.out.println(getMatch("heavy damages", pattern_1));
                System.out.println("################################");
                System.out.println(getMatch("damage", pattern_2));
                System.out.println(getMatch("no damage", pattern_2));
                System.out.println(getMatch("minor damage", pattern_2));
                System.out.println(getMatch("minor damages", pattern_2));
                System.out.println(getMatch("heavy damage", pattern_2));
                System.out.println(getMatch("heavy damages", pattern_2));

        }
}
```

Output:

```
match on #((minor|heavy)\s{1})?damage(s)?# for damage
no match on #((minor|heavy)\s{1})?damage(s)?# for no damage
match on #((minor|heavy)\s{1})?damage(s)?# for minor damage
match on #((minor|heavy)\s{1})?damage(s)?# for minor damages
match on #((minor|heavy)\s{1})?damage(s)?# for heavy damage
match on #((minor|heavy)\s{1})?damage(s)?# for heavy damages
################################
match on #((heavy )?damage|minor damages)# for damage
no match on #((heavy )?damage|minor damages)# for no damage
no match on #((heavy )?damage|minor damages)# for minor damage
match on #((heavy )?damage|minor damages)# for minor damages
match on #((heavy )?damage|minor damages)# for heavy damage
no match on #((heavy )?damage|minor damages)# for heavy damages
```

2 Objects

Define a class Student with the reference variables studentId and name, a constructor and default methods. It is given:
ID Name
0054 Albert Einstein
1234 Gottfried Wilhelm Leibniz
5421 Carl Friedrich Gauss

Write code for a console output (Name and ID) that is sorted by the student names.

Solution:

```
package testen.java;

public class Student
{
	private int id;
	private String name;
	public Student()
	{

	}
	public Student(int id,String name)
	{
		this.id = id;
		this.name = name;
	}

	public void setId(int id)
	{
		 this.id = id;
	}

	public int getId()
	{
		return id;
	}
	public void setName(String name)
	{
		this.name = name;
	}
	public String getName()
	{
		return this.name;
	}
```

```
	public String getFamilyName()
	{
		String[] fName = getName().split("\\s");
		return fName[fName.length-1];
	}

	public String toString()
	{
		String str = String.format("id:%06d name:%s",id,name);
		return str;
	}
}

package testen.java;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.stream.Collectors;
import java.util.stream.Stream;

@SuppressWarnings("rawtypes")
public class StudentList<Stundent> extends ArrayList
{
	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	@SuppressWarnings("unchecked")
	public void addStudent(Student s)
	{
		this.add(s);
	}

	@SuppressWarnings("unchecked")
	public void sort()
	{
		this.sort((s1, s2)->	{
	return ((Student)s1).getName().compareTo(((Student)s2).getName()); });
	}

	@SuppressWarnings("unchecked")
	public void reverseSort()
	{
		this.sort(Comparator.comparing(Student::getName).reversed());
	}

	@SuppressWarnings("unchecked")
	public void familyNameSort()
	{
		this.sort(Comparator.comparing(Student::getFamilyName));
	}
	public String displayList()
	{
```

```
            @SuppressWarnings("unchecked")
            Stream<Student> stream = this.stream();
      return stream.map(i->i.toString()).collect(Collectors.joining(" \n"));
      }

      @SuppressWarnings("unchecked")
      public static void main(String args[])
      {
            StudentList<Student> sList = new StudentList<Student>();
            sList.add(new Student(54,"Albert Einstein"));
            sList.add(new Student(1234,"Gottfried Wilhelm Leibniz"));
            sList.add(new Student(5421,"Carl Friedrich Gauss"));
            sList.add(new Student(123456,"Traian Geicu"));
            System.out.println("Before Sort\n"+sList.displayList());
            sList.sort();
            System.out.println("After Sort\n"+sList.displayList());
            sList.reverseSort();
            System.out.println("Reverse Sort\n"+sList.displayList());
            sList.familyNameSort();
            System.out.println("Family Name Sort\n"+sList.displayList());
      }
}
```

Output:

```
Before Sort
id:000054 name:Albert Einstein
id:001234 name:Gottfried Wilhelm Leibniz
id:005421 name:Carl Friedrich Gauss
id:123456 name:Traian Geicu
After Sort
id:000054 name:Albert Einstein
id:005421 name:Carl Friedrich Gauss
id:001234 name:Gottfried Wilhelm Leibniz
id:123456 name:Traian Geicu
Reverse Sort
id:123456 name:Traian Geicu
id:001234 name:Gottfried Wilhelm Leibniz
id:005421 name:Carl Friedrich Gauss
id:000054 name:Albert Einstein
Family Name Sort
id:000054 name:Albert Einstein
id:005421 name:Carl Friedrich Gauss
id:123456 name:Traian Geicu
id:001234 name:Gottfried Wilhelm Leibniz
```

3 Design pattern

What is a singleton pattern and when do you use it? Please use an example.

Singleton is a standard GOF pattern (creational pattern) used when is required just one instance of object which latter will be used by other class. Can be usefull as storage and could also be various deviation from pattern in order to have a limited numbers of instances. Eg. Connections to a database could be handled via a pool which will generate just up till a limited number of instances. A solution with additional explanations is written and implemented for the Robot problem

4 Inheritance

Look at the following class:

```
public class Product {
String name;
String description;
double price;
public Product ( String name, String desc, double price ) {
this.name = name;
this.description = desc;
this.price = price;
}
public final double getPriceWithTax() {
return price * 1.19;
}
public String toString() {
return name + " _ " + description + " _ " + price + " EUR";
}
}
```

Write a subclass Clothing that extends the class Product. The subclass must have additional attributes for the size (as int) and the material (as String). The new attributes shall be initialized in the constructor like the other attributes name, description and price, as well as implemented in the method toString() to override the output stream

Solution:

```
package testen.java;

public class Product
{
	private String name;
	private String description;
	private double price;
// could use also a  constructor with object as alternate way of initialization
// do not add getter and setter since want the force initialization via constructor

	public Product(Product p)
	{
		this.name = p.name;
		this.description = p.description;
		this.price = p.price;
	}
	public Product (String name, String desc, double price)
	{
		this.name = name;
		this.description = desc;
```

```
		this.price = price;
	}
	public final double getPriceWithTax()
	{
		return price * 1.19;
	}
	public String toString()
	{
		return name + " _ " + description + " _ " + price + " EUR";
	}
}


package testen.java;

public class Clothing extends Product
{
	private int size;
	private String material;

	//less verbose
	public Clothing (Product p, int size, String material)
	{
		super(p);
		this.size = size;
		this.material = material;
	}
	public Clothing (String name, String desc, double price, int size, String material)
	{
		super(name, desc, price);
		this.size = size;
		this.material = material;
	}

	public String toString()
	{
		return super.toString() + " _ size " + this.size + " _ material " + this.material;
	}

	public String productToString()
	{
		return super.toString();
	}
}
```

```java
package testen.java;

public class ClothingShop
{
	public void shop()
	{
		Clothing clothing =
new Clothing("T-Shirt", "bicolor with red and yellow XL",10,40,"80% cotton");

		System.out.println("#product_1# "+ clothing.productToString());
		System.out.format("%s _ PriceWithVat %.2f EUR \n",
clothing,clothing.getPriceWithTax());

		//other instantiation
		//since we linked clothing with a new product final modifier will be linked also with a
new product
		//so it's reflect properly the new item price
		Product product =
new Product("T-Shirt", "bicolor with blue and white XXL",30);
		clothing = new Clothing(product, 30,"100% cotton" );
		System.out.println("#product_2# "+product);
	//	System.out.println("#clothing# "+clothing);
		//2 decimal prices
	System.out.format("%s _ PriceWithVat %.2f EUR", clothing,clothing.getPriceWithTax());


	}

	public static void main(String args[])
	{
		new ClothingShop().shop();
	}
}
```

```
Output:

#product_1# T-Shirt _ bicolor with red and yellow XL _ 10.0 EUR
T-Shirt _ bicolor with red and yellow XL _ 10.0 EUR _ size 40 _ material 80%
cotton _ PriceWithVat 11.90 EUR
#product_2# T-Shirt _ bicolor with blue and white XXL _ 30.0 EUR
T-Shirt _ bicolor with blue and white XXL _ 30.0 EUR _ size 30 _ material
100% cotton _ PriceWithVat 35.70 EUR
```

5 Threads
Imagine a simple robot that can randomly move forward and backward as well as left and right.
The task is to show the movement of the robot in the form of a console output (e.g.,
System.out.println("left..."), System.out.println("right..."), ...). Define two
classes HorizontalThread and VerticalThread as threads. The class HorizontalThread
process the robots movement left and right, the class VerticalThread process the robots
movement forward and backward. The movements have to be displayed in the console. Now,
implement a class Robot that starts both threads.

```java
package testen.java;
/*
 * 1.used RobotThread class as extension base for our threads
 * this in order to get the DRY principle
 * (not to write twice the same methods with same functionalities)
 * 2.adding some customizations via constructor and setter
 * 3.robot will run indefinitely with no args constructor
 * 4.robot will run till step with a delay for specific args on constuctor
 * 5.data structure based on singleton is used to track move and latter display all
 * (type of list that allow to maintain the order of insertion)
 * 6. by uncomment System.out on run method could see step by step moves
 */
public class RobotThread extends Thread{

	private boolean flag = true;
	private String move = "none";
	private String [] moves = {"move_1", "move_2"};
	int totalMoves = -1;
	int moveTime = 100;
	public RobotThread()
	{

	}

	public RobotThread(int totalMoves, int moveTime)
	{
		this.totalMoves = totalMoves;
		this.moveTime = moveTime;
	}
```

```
	public void run()
	{
		while(flag)
		{
			double choice = 2*Math.random();
			move = moves[1];
			if(choice<1) move = moves[0];
			//System.out.println(move+":"+System.currentTimeMillis());
			SingletonQueue.getInstance().addMove(move+":"+System.currentTimeMillis());
			totalMoves--;
			if(totalMoves == 0)
				flag = false;


		}
	}
	public void setMoves(String move_1, String move_2)
	{
		moves[0] = move_1;
		moves[1] = move_2;
	}

}


package testen.java;

public class HorizontalThread extends RobotThread
{
	public HorizontalThread()
	{
		super();
		super.setMoves("left", "right");
	}

	public HorizontalThread(int counter, int delay)
	{
		super(counter,delay);
		super.setMoves("left","right");
	}
}
```

```java
package testen.java;

public class VerticalThread extends RobotThread
{
	public VerticalThread()
	{
		super();
		super.setMoves("forward", "backwark");
	}

	public VerticalThread(int counter, int delay)
	{
		super(counter,delay);
		super.setMoves("forward","backward");
	}
}
```

```java
package testen.java;

import java.util.LinkedList;
import java.util.stream.Stream;
/*
 * we could use a particular data structure to track moves and latter list them all
 * (need one and the same instance:singleton for RobotThead:track and Robot:display)
 */
public class SingletonQueue extends LinkedList<String>
{

	private static final long serialVersionUID = -951747679375814219L;
	private static SingletonQueue sq = null;
	private SingletonQueue()
	{

	}

	public static SingletonQueue getInstance()
	{
		if(sq == null) sq = new SingletonQueue();
		return sq;
	}

	public void addMove(String s)
	{
		synchronized (sq)
		{
			sq.add(s);
		}
	}
```

```
	public String getMoveAndRemove()
	{
		if(!sq.isEmpty())
		{
			return sq.poll();
		}
		return "no_move";
	}

	public void showList()
	{
		System.out.println("no. rec="+ sq.size());
		System.out.println("moves:time");
		Stream<String> s = sq.stream();
	   s.forEach(System.out::println);
	}
}

package testen.java;
/*
 * 1.this is the running class for robot
 * 2.association on classes and global logic is here
 * (specific logic also on RobotThread : Moves)
 * 3.since we could display on the end all the moves need also
 * to wait on main thread that both HT and VT to finsh run method
 * in order to have the singleton with all records
 * 4. java.io may cause delay on display so an accurate response will be
 * with on reading from our list (and not when move)
 */
public class Robot
{
	public  void runRobot() throws InterruptedException
	{
		// Thread first  param = number of moves
		// Thread second param = time to move in milis
		HorizontalThread ht = new HorizontalThread(20,30);
		VerticalThread vt = new VerticalThread(10,100);
		ht.start();
		vt.start();
		//need to wait till both thread finsh run method
		while(ht.isAlive() || vt.isAlive())
		{
			Thread.sleep(100);
		}
		SingletonQueue.getInstance().showList();
	}
```

```
	public static void main(String args[]) throws InterruptedException
	{
		new Robot().runRobot();

	}
}
```

Output:

```
no. rec=30
moves:time
backward:1522855393666
right:1522855393666
backward:1522855393667
forward:1522855393667
left:1522855393667
backward:1522855393667
forward:1522855393667
right:1522855393667
backward:1522855393667
left:1522855393667
forward:1522855393667
right:1522855393667
backward:1522855393667
left:1522855393667
backward:1522855393667
left:1522855393667
right:1522855393667
backward:1522855393667
right:1522855393667
right:1522855393667
left:1522855393667
left:1522855393667
right:1522855393667
right:1522855393667
left:1522855393667
left:1522855393667
right:1522855393667
right:1522855393667
left:1522855393667
left:1522855393667
```

6 :Correct the code
1: public class carThread implements Thread {
2: final String brand;
3: final String model;
4: final double price;
5:
6: public carThread (String brand, String model) {
7: this.brand = brand;
8: this.model = model;
9: }
10:
11: public void run() {
12: while(true) {
13: System.out.println(“hello my name is this.brand”);
14: Thread.sleep(300);
15: }
16: }
17:
18: public static void main(String[] args) {
19: new carThread(“Audi”).run();
20: new carThread(“BMW”).run();
21: System.out.println(“carThreads are running... “);
22: }
23:
What's wrong with this class? Correct all errors.
Solution:

```
package testen.java;
/*
 * 1.we use alternative way for thread by implementing run method
 * 2.if invoke run directly the behavior it will be executed in MainThread
method
 * and not in our thread (no errors on compile but not the wanted behavior)
 * 3. name class could be even carThread but it's custom to capitalize
FistLetter
 */
public class CarThread implements Runnable
{
	final String brand;
	final String model;
	final double price = 0;

	// since price is already initialized and it's final it cannot be
changed
	// brand and model are just declared but not initialized
	// could remove final in order to be able to alter the price
	public CarThread(String brand, String model)
	{
		this.brand = brand;
		this.model = model;

	}
```

```
	public void run()
	{
		while(true)
		{
System.out.println("hello my name is " + this.brand + " "+ this.model + " and
worths zero: " + this.price +" euros");
			try
			{
				Thread.sleep(300);
			}
			catch(InterruptedException e)
			{
				System.out.println(e);
			}
		}
	}

	public static void main(String[] args)
	{
		new Thread(new CarThread("Audi","8")).start();
		new Thread(new CarThread("BMW","i8")).start();
		new Thread(new CarThread("Audi","7")).start();
		System.out.println("carThreads are running... ");
	}

}
```

Output:

```
carThreads are running...
hello my name is Audi 8 and worths zero: 0.0 euros
hello my name is Audi 7 and worths zero: 0.0 euros
hello my name is BMW i8 and worths zero: 0.0 euros
hello my name is Audi 7 and worths zero: 0.0 euros
hello my name is Audi 8 and worths zero: 0.0 euros
hello my name is BMW i8 and worths zero: 0.0 euros
hello my name is Audi 7 and worths zero: 0.0 euros
hello my name is Audi 8 and worths zero: 0.0 euros
hello my name is BMW i8 and worths zero: 0.0 euros
```

7 Bonus

Write a method with only one boolean parameter. Depending on the parameter the method has to return “a”, “b”, or “c”.

Solutions:

```
package testen.java;
public class BonusTest
{
	@SuppressWarnings("null")
	/*
	 * A:True
	 * B:False
	 * C:Null
	 */
	public static void main(String args[])
	{
		InvokeMethod im = new BonusTest().new InvokeMethod();
		//solution 1 with primitive but with 3rd result outside method
		System.out.println("Solution 1");
		System.out.println(im.InvokePrimitive(true));
		System.out.println(im.InvokePrimitive(Boolean.TRUE));
		System.out.println(im.InvokePrimitive(false));
		System.out.println(im.InvokePrimitive(Boolean.FALSE));
		//there is no Boolean.NULL so can assign obj to be
		Boolean obj = null;
		try
		{
			System.out.println(im.InvokePrimitive(obj));
		}
		catch(NullPointerException e)
		{
			//this is the case when it's possible to pass a wrong param
			// but this will not go inside method so we cannot catch
there
			// primitiveMethod return only 2values the third is outside
method
			System.out.println("C");
		}

		//Solution 2
		// use the obj Boolean who can trace all 3 entries :true, false,
null
		// this will do our purpose but the parameter should be Object
Boolean and not primitive
		System.out.println("Solution 2");
		System.out.println(im.InvokeObject(true));
		System.out.println(im.InvokeObject(Boolean.TRUE));
		System.out.println(im.InvokeObject(false));
		System.out.println(im.InvokeObject(Boolean.FALSE));
		System.out.println(im.InvokeObject(obj));


	}
```

```
	private class InvokeMethod
	{
		public String InvokePrimitive(boolean bool)
		{
			if(bool == true) return "A";
			return "B";
		}
		public String InvokeObject(Boolean bool)
		{
			if(bool == null) return "C";
			else if(bool == true) return "A";
				return "B";
		}
	}
}
```

Output:

```
Solution 1
A
A
B
B
C
Solution 2
A
A
B
B
C
```

Thank You

Traian GEICU