Lab 6

1.

class BulkDiscount extends DiscountBasic {

int minimum;

double percentOff;

public BulkDiscount(int count, double itemCost){

minimum = count;

percentOff = itemCost;

}

public double getDiscount(int count, double itemCost){

if (count < minimum){

return 0;

}

else{

return itemCost \* count \* (percentOff / 100.0);

}

}

}

2.

class VIPDiscount extends DiscountBasic{

int minAmount;

double percentOff;

public VIPDiscount(int count, double itemCost){

minAmount = count;

percentOff = itemCost;

}

public double getDiscount(int count, double itemCost){

if (count \* itemCost < minAmount){

return 0;

}

else{

return itemCost \* count \* (percentOff / 100.0);

}

}

}

6.

public static void printSleep(Sleep[] s){

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

System.out.println(s[i].sleep());

}

}

Lab 7

1.

class Person implements Comparable<Person> {

protected String name;

protected int age;

public int compareTo(Person p){

if(this.age > p.age)

return 1;

else if(this.age < p.age)

return -1;

else

return 0;

}

public Person(String name, int age) {

this.name = name;

this.age = age;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String toString() {

return String.format("%s(%d)", name, age);

}

}

2.

abstract class Shape implements Comparable<Shape> {

protected Point p; //top-left corner

public int compareTo(Shape s){

return this.perimeter() - s.perimeter();

}

public Shape() {

p = new Point();

}

public Shape(int x, int y) {

p = new Point(x,y);

}

public String toString() {

return String.format("%d", perimeter());

}

public abstract int perimeter();

}

3.

class Outer {

private static int count = 0;

public static int getCount() { return count; }

public void sample() {

Nested n1 = new Nested();

n1.increment();

}

static class Nested{

public int increment(){

return count++;

}

}

4.

public static Point highestPoint(List<Point> points) {

Iterator<Point> pointIterator = points.iterator();

int highest = 0;

Point highestPoint = null;

// TODO use the iterator to complete this method!

int number = 0;

while (pointIterator.hasNext()) {

if(highest < points.get(number).y) {

highestPoint = points.get(number);

highest = points.get(number).y;

}

number++;

pointIterator.next();

}

return highestPoint;

}

6.

public String riskyMethodHandler(){

try{

riskyMethod();

}

catch(NullPointerException e){

return "java.lang.NullPointerException";

}

catch(Exception e){

return "Exception!";

}

return "No exception!";

}

Lab 8

1.

public void match(String s) {

String p = "^[a-z]{3,4}\\d{3}$";

System.out.println(s.matches(p)? "match " + s: "does not match " + s);

}

2.

public void match(String s) {

String p = "^\\d{3}\\-\\d{2}\\-\\d{4}$";

System.out.println(s.matches(p)? "match " + s: "does not match " + s);

}

3.

public void match(String s) {

String p = "^[A-Z]{1}[a-z]+(\\s[A-Z]{1}[a-z]+)\*$";

System.out.println(s.matches(p)? "match " + s: "does not match " + s);

}

4.

public void match(String s) {

String p = "^[A-Z]{2}\\d{6}$";

System.out.println(s.matches(p)? "match " + s: "does not match " + s);

}

5.

public void match(String s) {

String p = "^\\d{3}-[A-Z]{3}\\d{1}|\\d{3}[A-Z]{1}-[A-Z]{1}\\d{2}|\\d{3}\\-\\d{3}$";

System.out.println(s.matches(p)? "match " + s: "does not match " + s);

}