Student ID: .....

TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI

VICTORIA
UNIVERSITY OF WELLINGTON

EXAMINATIONS – 2014
TRIMESTER 1

SWEN221

Software Development

Time Allowed: THREE HOURS

Instructions: Closed Book.
There are 180 possible marks on the exam.

Answer all questions in the boxes provided.
Every box requires an answer.
If additional space is required you may use a separate answer booklet.

No calculators permitted.
Non-electronic Foreign language dictionaries are allowed.

No reference material is allowed.

Question	Topic	Marks
1.	Debugging and Code Comprehension	30
2.	Java Masterclass	30
3.	Interfaces & Cloning	30
4.	Exceptions	30
5.	Testing	30
6.	Generics	30
	Total	180

Question 1. Debugging and Code Comprehension [30 marks]

Consider the following classes, which compile without error:

```
1  // A square on the board
2  abstract class Square {
3     public abstract void attack();
4  }
5
6  // A blank square on the board
7  public class Blank extends Square {
8     public void attack() {}
9  }
10
11 // A monster on the board
12 public class Monster extends Square {
13     private int hitPoints;
14
15     public Monster(int hitPoints) { this.hitPoints = hitPoints; }
16
17     public void attack() { hitPoints --; }
18
19     public boolean isDestroyed() { return hitPoints == 0; }
20 }
21
22 // The board
23 public class Board {
24
25     // A width * height grid of squares.  Each square
26     private Square[][] squares;
27
28     public Board(int width, int height) {
29        squares = new Square[width][height];
30     }
31
32     public void place(Monster m, int x, int y, int width) {
33         for(int i=x;i!=width;++i) {
34           squares[i][y] = m;
35         }
36     }
37
38     public void attack(int x, int y) { squares[x][y].attack(); }
39 }
```

(a) Based on the code given on page 2, state the output you would expect for each of the following code snippets:

(i) [2 marks]

```
1 Board board = new Board(10,10);
2 Monster m = new Monster(5);
3 board.place(m,0,0,3);
4 board.attack(0,0);
5 System.out.println(m.isDestroyed());
```

```
false
```

(ii) [2 marks]

```
1 Board board = new Board(10,10);
2 Monster m = new Monster(2);
3 board.place(m,0,0,3);
4 board.attack(1,0);
5 board.attack(2,0);
6 System.out.println(m.isDestroyed());
```

```
true
```

(iii) [2 marks]

```
1 Board board = new Board(10,10);
2 Monster m = new Monster(2);
3 board.place(m,0,0,3);
4 board.place(m,0,5,3);
5 board.attack(1,0);
6 board.attack(2,5);
7 System.out.println(m.isDestroyed());
```

```
true
```

(iv) [2 marks]

```
1 Board board = new Board(10,10);
2 Monster m = new Monster(5);
3 board.place(m,0,0,3);
4 board.attack(1,5);
5 System.out.println(m.isDestroyed());
```

```
Exception in thread "main" java.lang.NullPointerException
```

(b) [5 marks] Rewrite the `Board` constructor so that it initialises every square in `squares` to a `Blank` square.

```
public Board(int width, int height) {
    squares = new Square[width][height];
    for(int x = 0; x < width; ++x) {
        for(int y = 0; y < height; ++y) {
            squares[x][y] = new Blank();
} } }
```

(c) [3 marks] Consider the method `Monster.attack()`. Does this *overload* or *override* the method `Square.attack()`? Justify your answer.

`Monster.attack()` overrides `Square.attack()` because it is defined in a subclass, and has the same name a signature (i.e. the same name and parameter types).

(d) [3 marks] The class `Square` is declared as **`abstract`**. Briefly, discuss what this means.

This means that the class itself cannot be instantiated. Instead, only subclasses which implement its abstract method `attack()` can be instantiated. Furthermore, all concrete subclasses must implement this method.

(e) Two squares on the board may refer to the same `Monster`.

(i) [2 marks] Briefly, explain the meaning of this in terms of *objects* and *references*.

This means that two elements in the array `Board.squares` may reference the same instance of `Monster` (i.e. the same object).

(ii) [5 marks] Briefly, discuss what effect this has on how the program works.

This means that a `Monster` object can span multiple squares of the board. Furthermore, attacking any one of these squares will affect the same `Monster` object. Note also that it means a single `Monster` object can occupy squares which are not adjacent to each other. This can be achieved by calling the `place()` method multiple times with the same reference, and may not have been an intended behaviour.

(f) [4 marks] Two squares on the board may refer to the same `Blank` square. Briefly, discuss why this does not affect how the program works.

This does not affect how the program works because `Blank` objects have no state (they are immutable). Therefore, it makes no difference whether two squares refer to the same `Blank` object or not.

Question 2. Java Masterclass [30 marks]

As for the self assessment tool, for each of the following questions, provide in the answer box the code that should replace `[???]`.

(a) [4 marks]

```
1  //The answer must have balanced parentesis
2  public class Exercise{
3    public static void main(String [] arg){
4      int foo=10;
5      assert (10==[???]);
6      assert (11==[???]);
7      assert (12==[???]);
8      assert (13==[???]);
9    }
10 }
```

foo++

(b) [4 marks]

```
1  //The answer must have balanced parenthesis,
2  class Avatar{
3    Avatar(String name){this.name=name;}
4    String name;
5  }
6  class NintendoAvatar extends Avatar{[???]}
7
8  public class Exercise{
9    public static void main(String [] arg){
10     assert (new NintendoAvatar().name.equals("Mario"));
11     assert (new NintendoAvatar("Luigi").name.equals("Luigi"));
12 }  }
```

```
NintendoAvatar() { super("Mario"); }
 NintendoAvatar(String name) { super(name); }
```

(c) [4 marks]

```
1  //The answer must have balanced parenthesis
2  class Base1{ int m(){return 1;}}
3  class Base2{ int m(){return 2;}}
4  class C1 extends Base1{ int m(){[???]}}
5  class C2 extends Base2{ int m(){[???]}}
6  public class Exercise{
7    public static void main(String [] arg){
8      assert new C1().m()==10;
9      assert new C2().m()==20;
10 }  }
```

```
return super.m()*10;
```

(d) [4 marks]

```
1  //The answer must have balanced parentesis
2  import java.util.HashSet;
3  class Elem { [???] }
4  public class Exercise{
5    public static void main(String [] arg){
6      HashSet<Elem> es=new HashSet<Elem>();
7      es.add(new Elem());
8      es.add(new Elem());
9      es.add(new Elem());
10     assert es.size()==1;
11 }  }
```

```
public int hashCode() { return 1;}
public boolean equals(Object obj) { return true; }
```

(e) [4 marks]

```
1  //The answer must have balanced parenthesis
2  class A{
3    int m(){return 1;}
4  }
5  public class Exercise{
6    public static void main(String [] arg){
7      A a=[???];
8      assert a.m()==2;
9    }
10 }
```

```
new A(){int m(){return 2;}}
```

(f) [10 marks]

```
1  //The answer must have balanced parenthesis
2  import java.util.ArrayList;
3  interface A{int m();}
4  public class Test {
5    public static void main(String[] arg){
6      ArrayList<A> a=new ArrayList<A>();
7      for(int i=0;i<10;i++){add(a);}
8      assert a.get(0).m()==0;
9      assert a.get(1).m()==1;
10     assert a.get(7).m()==7;
11     assert a.get(9).m()==9;
12   }
13   [???]
14 }
```

Hint: since `add()` is called from `main()` but is not declared, you may want to declare it.

```
static class AA implements A{
    int f; AA(int f){this.f=f;}
    public int m(){return f;}
}
static void add(ArrayList<A> a){a.add(new AA(a.size())); }
```

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Question 3. Interfaces & Cloning [30 marks]

(a) Consider the following classes and interfaces:

```
1  interface Shape {
2      boolean contains(int x, int y);
3      Shape clone();
4  }
5
6  public class Rectangle implements Shape {
7    private int x1;
8    private int y1;
9    private int x2;
10   private int y2;
11
12   public Rectangle(int x1, int y1, int x2, int y2) {
13     this.x1 = x1; this.y1 = y1;
14     this.x2 = x2; this.y2 = y2;
15   }
16   public bool contains(int x, int y) {
17     // Check x,y is contained within this rectangle
18     return x >= Math.min(x1,x2) &&
19            x <= Math.max(x1,x2) &&
20            y >= Math.min(y1,y2) &&
21            y <= Math.max(y1,y2);
22   }
23   public Shape clone() { [???] }
24 }
```

(i) [3 marks] Give an appropriate implementation of `clone()` for the `Rectangle` class.

```
return new Rectangle(x1,y1,x2,y2);
```

(ii) [5 marks] Briefly, discuss why there is no difference between a *deep clone* and a *shallow clone* for the `Rectangle` class.

- A shallow clone copies all data in a given object, but does not clone objects which it references
- A deep clone copies all data in a given object, and recursively clones objects which are referenced
- `Rectangle` contains only fields of primitive (i.e. none of reference type). Primitives are immutable values and cannot be cloned per se.

(b) Consider the following implementation of shape:

```
1  class ShapeUnion implements Shape {
2      private Shape[] shapes;
3
4      public ShapeUnion(Shape[] ss) {
5          this.shapes = ss;
6      }
7
8      public boolean contains(int x, int y) {
9          for(Shape s : shapes) {
10             if(s.contains(x,y)) { return true; }
11         }
12         return false;
13     }
14
15     public Shape clone() { [???] }
16 }
```

(i) [7 marks] Give an implementation of `clone()` for the `ShapeUnion` class which implements a *deep clone*. You may assume that a `Shape` cannot contain itself.

```
Shape[] nShapes = new Shape[shapes.length];
for(int i=0;i!=shapes.length;++i) {
    nShapes[i] = shapes[i].clone();
}
return new ShapeUnion(nShapes);
```

(ii) [5 marks] Suppose that a `Shape` was permitted to contain itself. Briefly, discuss how you would alter your `clone()` method to handle this.

The essential problem here is that cloning a `Shape` which contains itself will lead to an infinite loop and, eventually, a `StackOverflowException`. Dealing with this is surprisingly difficult, and require a way to record which instances of `Shape` have been previously visited. The algorithm is similar (in some ways) to how a depth-first search works. One approach to recording which objects have been visited is to use an `IdentityHashMap`.

(c) [5 marks] Consider again the constructor for `ShapeUnion`:

```
1       public ShapeUnion(Shape[] ss) {
2           this.shapes = ss;
3       }
```

This constructor assigns the `ss` parameter directly to the `shapes` field. Briefly, discuss whether you think this is a good or bad idea and what (if anything) you would do differently.

There are several problems with this constructor:

- Firstly, the constructor does not check the `ss` variable against **`null`**. Since this is eventually used within the `contains()` method without any check, we can assume is should not **`null`**.
- Secondly, the constructor assigns the array reference held in `ss` directly to the `shapes` field. This can cause problems if the calling object retains a reference to this array, as it could subsequently modify it and this would affect the `ShapeUnion` object.

(d) [5 marks] An `InverseShape` contains a `Shape` and includes all those points *not* in the contained `Shape`. Give an implementation for `InverseShape`.

```
class InverseShape implements Shape {
    private Shape shape;
    public InverseShape(Shape shape) {
        this.shape = shape;
    }
    public boolean contains(int x, int y) {
        return !shape.contains(x,y);
    }
    public Shape clone() {
        return new InverseShape(shape.clone());
    }
}
```

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Question 4. Exceptions [30 marks]

A New Zealand supermarket chain has a data management system which is able to recover employers data. This is their `getData()` method

```
1 public Employer getData(int id){
2   Employer result=null;
3   DBConnection db=new DBConnection("...");
4   Result r=db.query("select␣..."+id+"...");
5   // if a result comes back, just return it
6   if(r.size()==1){result=new Employer(r);}
7   // otherwise, must have been an invalid id.
8   return result;
9 }
```

(a) [2 marks] How does the current implementation handle the case of an invalid employer ID?

The current implementation returns `null`

(b) [5 marks] How would you modify this method in order to provide a better behaviour in the case of an invalid employer ID? Write down the new code for method `getData()`.

```
... if(r.size()==1){result=new Employer(r);}
 else throw new InvalidEmployerId(id);
```

(c) [4 marks] Write down the complete source code of any exception class that you would define for your answer to (b).

```
class InvalidEmployerId extends RuntimeException{
  int id;
  InvalidEmployerId(int id){this.id=id;}}
```

(d) [5 marks] Class `DBConnection` offers a method `close()`. The current implementation does not close the database connection. How would you modify this method in order to ensure the database connection is closed?

```
DBConnection db=null;
try{
  db=new DBConnection("...");
  Result r=db.query("select ..."+id+"...");
  if(r.size()==1){result=new Employer(r);}
  else return new InvalidEmployerId(id);
}finally{if(db!=null)db.close();}
```

(e) `Throwable` is the common supertype for all Java exceptions.

(i) [2 marks] Is `Throwable` checked or unchecked?

Must be checked, as `Throwable :> Exception :> RuntimeException`

(ii) [6 marks] Explain why it must be checked / unchecked and why it could not be otherwise. You are encouraged to use a code example.

if `throwable` was unchecked, then it could be possible to trick the Java type system, as in the following:

```
throw (Throwable)new MyCheckedException();
```

(iii) [6 marks] The `Exception` class has the following constructor:

```
Exception(String message, Throwable cause)
```

Describe the meaning of both parameters. In particular, when is the second parameter useful?

The first parameter should encode a human readable description of the error, while the second (optional) parameter should contain an other exception, that have conceptually caused the new one.
This is used in many cases where there is a concept of delegation of responsibility. We can see it in the Java reflection, where if the `invoke`d method generate an exception, the corresponding `InvocationTargetException` receive such exception as its cause.

SPARE PAGE FOR EXTRA ANSWERS

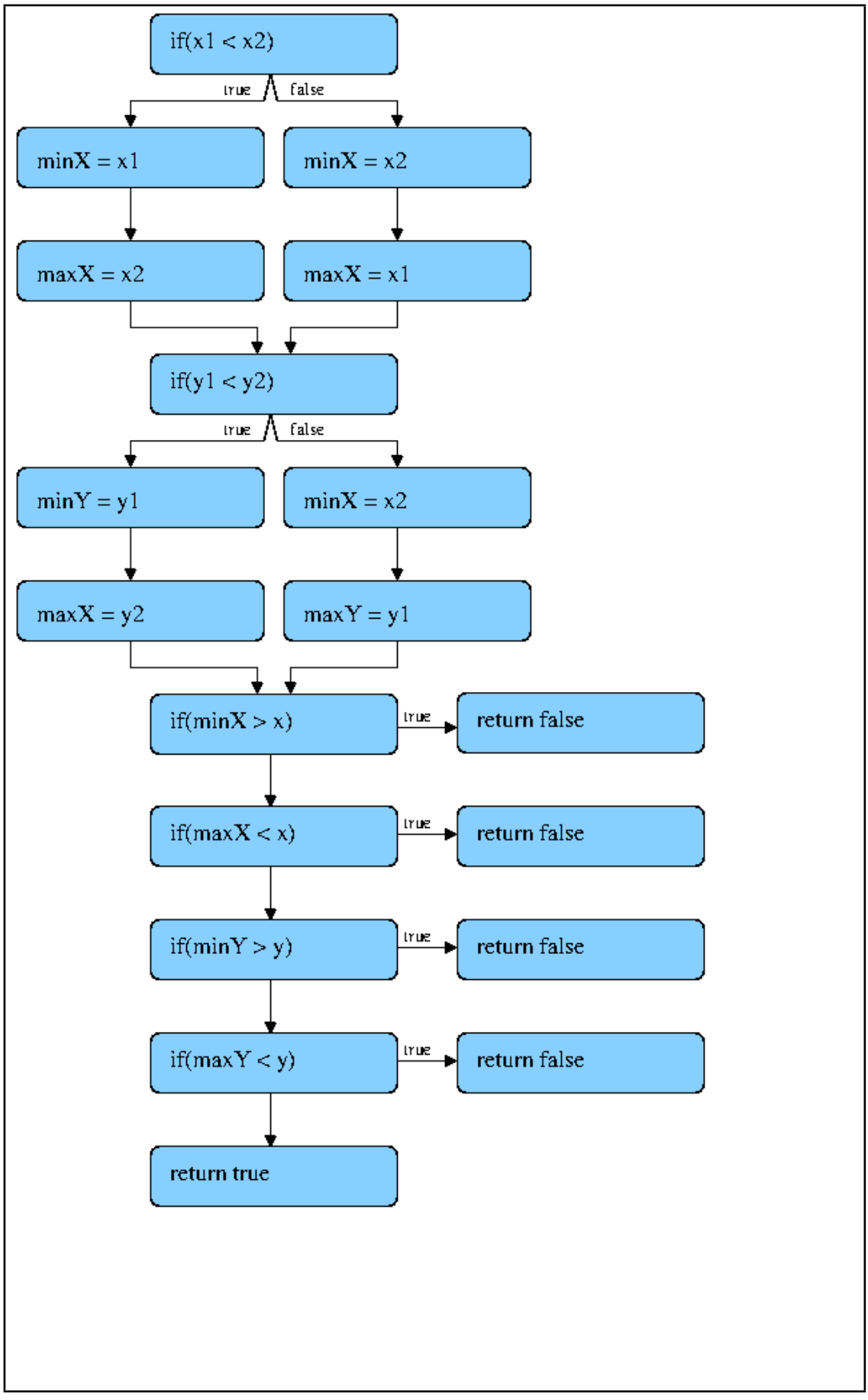
Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Question 5. Testing [30 marks]

(a) Consider the following classes which compile without error:

```
1  public class Rectangle {
2    private int x1;
3    private int y1;
4    private int x2;
5    private int y2;
6
7    public Rectangle(int x1, int y1, int x2, int y2) {
8      this.x1 = x1; this.y1 = y1;
9      this.x2 = x2; this.y2 = y2;
10   }
11
12   public boolean contains(int x, int y) {
13     int minX;
14     int maxX;
15     int minY;
16     int maxY;
17     // Determine minimum and maximum bounds
18     if(x1 < x2) { minX = x1; maxX = x2; }
19     else {
20       minX = x2; maxX = x1;
21     }
22     if(y1 < y2) { minY = y1; maxY = y2; }
23     else {
24       minY = y2; maxY = y1;
25     }
26     // Check whether point x,y is contained
27     if(minX > x) { return false; }
28     if(maxX < x) { return false; }
29     if(minY > y) { return false; }
30     if(maxY < y) { return false; }
31     return true;
32 } }
33
34 public class RectangleTests {
35   @Test void testContains_1() {
36     assertTrue(new Rectangle(0,0,5,5).contains(1,1));
37   }
38   @Test void testContains_2() {
39     assertTrue(new Rectangle(5,5,0,0).contains(1,1));
40   }
41   @Test void testContains_3() {
42     assertFalse(new Rectangle(0,0,5,5).contains(-1,1));
43   }
44   @Test void testContains_4() {
45     assertFalse(new Rectangle(0,0,5,5).contains(6,1));
46 } }
```

(i) [8 marks] Draw the *control-flow graph* for the `Rectangle.contains(int,int)` method:



(ii) [2 marks] What is *statement coverage*?

The proportion of statements covered by the tests

(iii) [2 marks] Give the total *statement coverage* of class `Rectangle` obtained from the tests in `RectangleTests`.

21 statements are covered out of 23, which is 91%

(iv) [2 marks] What is *branch coverage*?

The proportion of branching statements where both sides of the branch are tested

(v) [2 marks] Give the total *branch coverage* of class `Rectangle` obtained from the tests in `RectangleTests`.

4 branches are covered out of 6, which is 67%

(b) The *path coverage* criterion counts the proportion of all possible execution paths which are tested.

(i) [3 marks] Give the total number of possible execution paths through the method `Rectangle.contains()`.

There are $2 * 2 * 5 = 20$ possible execution paths.

(ii) [2 marks] Give the total *path coverage* of class `Rectangle` obtained from the tests in `RectangleTests`.

There are only 4 out of 20 execution paths tested, which is 20%

(iii) [4 marks] Give two additional test cases which increase the path coverage obtained for `Rectangle`.

```
@Test void testContains_5() {
    assertTrue(new Rectangle(5,0,0,5).contains(1,1));
}
@Test void testContains_6() {
    assertTrue(new Rectangle(0,5,5,0).contains(1,1));
}
```

(iv) [2 marks] Briefly, describe what an *infeasible path* is.

An infeasible path is an execution path through a function which cannot, in fact, be taken. This is because the logic of the function prevents this particular path from being possible.

(v) [3 marks] Why is path coverage impossible to measure in general?

Because, in the presence of loops and polymorphism, there are potentially an infinite number of possible execution paths.

Question 6. Generics [30 marks]

Consider the following code

```
1  import java.util.ArrayList;
2
3  class Point{
4    int x;  int y;
5    Point(int x, int y){ this.x=x;  this.y=y; }
6  }
7  class ColoredPoint extends Point{
8    int color;
9    ColoredPoint(int x, int y,int color) {
10     super(x, y); this.color=color;
11   }
12 }
```

(a) [5 marks] There are many possible representations for colours. The class `ColoredPoint` uses an **`int`**. Write instead a generic class `GenericPoint<T>` that uses any kind of type as a representation of a colour.

```
class GenericPoint<T> extends Point{
   T color;
   GenericPoint(int x, int y,T color) {
     super(x, y); this.color=color;
   }
 }
```

(b) Consider the following code

```
1  public class GenericTest {
2    static void m(ArrayList<Point> p){
3     [???]//you will be asked to fill the hole here
4    }
5
6    public static void main(String[] args){
7      ArrayList<ColoredPoint> cps=new ArrayList<ColoredPoint>();
8      try{
9        m((ArrayList<Point>)(Object)cps);
10     }
11     catch(Throwable t){}
12     for(ColoredPoint p:cps){
13       System.out.println(p.color);
14     }
15   }
16 }
```

Initially, Bob the programmer tried to pass variable `cps` directly to the method `m()`, but this caused a compilation error; he could not understand the reason for such an error, thus he decided to trick the type system and cast the error away (line 9).

(i) [5 marks] Explain the effect of the two casts in line 9, i.e. what happens when `m((ArrayList<Point>)(Object)cps);` is executed.

First cps is casted to `Object`, and then the result is casted to `ArrayList`. Casts are just controls, they do not modify the state of the program. In addition, all the generic informations are ignored.
In this particular case the generic information is `Point` on the cast and `ColoredPoint` on cps.
The system control that the object referred by `cps` is indeed a valid `ArrayList`, ignoring the different generic annotations.
That is, there is no control at all on the generic type, but only on the raw one.

(ii) [7 marks] Inserting such casts is unsafe! Provide an example implementation of the method `m()`, (replacing the `[???]` sign) that forces the method `main()` to throw an exception.

The expected solution was `p.add(new Point(1, 2));`
However, also `p.add(null);` do the job.

(iii) [8 marks] In your own words, explain why an exception was thrown above.

Thanks to generic type erasure and casts is possible to force Java to put values of incorrect types inside collections. Such value is than casted to the expected type when leaving the collection, so is possible to get a `ClassCastException` whenever a value is extracted by a collection.

(iv) [5 marks] What Bob really wanted, was to allow the method `m()` to take both `ArrayList<ColoredPoint>` and `ArrayList<Point>`. Write down a suitable generic method signature for method `m()`, so that both the following calls would be accepted:

```
1 m(new ArrayList<ColoredPoint>());
2 m(new ArrayList<Point>());
```

but the following would be rejected

```
1 m(new ArrayList<String>());
```

```
static void m(ArrayList<?  extends Point> p)
```

* * * * * * * * * * * * * * *

Student ID: .....

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Student ID: .....

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.