

CIS 120

Midterm II — October 24, 2008

Name: _____

Student ID: _____
(from your PennCard)

Email _____ @seas.upenn.edu

Lab:(circle one)

201 202 203 204 205 206
W 3:00 PM W 12:00 PM W 4:00 PM R 10:00 AM R 3:00 PM R 4:00 PM

- This exam text and attached code, the *Java Backpack Reference Guide*, and your brain are the only information sources you can use during this exam.
- You have 50 minutes to answer all of the questions. The entire exam is worth 50 points. The point value of each question is given to help you plan your time.
- Write your answers on the exam pages. The back side of each page may be used as a scratch pad.
- The answers to the questions require only basic Java language facilities and the code and APIs given in the appendix. No other classes or packages are needed.
- Questions during the exam should be about the wording of the exam only. If you have a question, raise your hand and we'll come to you. (This is less disruptive for others than if you come to us.)
- **Partial credit will be given where appropriate.**
- DON'T PANIC! If you find a question that you cannot solve right away, consider moving on and returning to it after you finish the other questions.

	Score	Max
1		12
2		12
3		8
4		18
Total		50

1. (12 points) The first question (on the next page) concerns the code below:

```
static class MyException extends Exception {}

static void m1 () throws MyException {
    throw new MyException();
}

static void m2 () {
    throw new RuntimeException ();
}

static int m3 (Object o1, Object o2) throws MyException {
    if (o1.equals(o2)) {
        m1();
        return 0;
    } else {
        return 1;
    }
}

static int m4 (Object o1, Object o2) {
    try {
        m2();
        m3(o1, o2);
        return 0;
    } catch (MyException e) {
        return 1;
    }
}

static int m5 (Object o1, Object o2) {
    try {
        m3(o1, o2);
        return 0;
    } catch (Exception e) {
        return 1;
    }
}

static int m6 (Object o1, Object o2) throws MyException {
    try {
        m3(o1, o2);
        return 0;
    } catch (MyException e) {
        throw e;
    } catch (RuntimeException e) {
        return 1;
    }
}
}
```

This question has four parts about the code on the previous page:

(a) Circle all possible results from a call to method m3 (with any arguments):

NullPointerException MyException RuntimeException

Answer: NullPointerException, MyException, 1

(b) Circle all possible results from a call to method m4 (with any arguments):

NullPointerException MyException RuntimeException

Answer: RuntimeException

(c) Circle all possible results from a call to method m5 (with any arguments):

NullPointerException MyException RuntimeException

Answer: 1, 0

(d) Circle all possible results from a call to method m6 (with any arguments):

NullPointerException MyException RuntimeException

Answer: 1, 0, MyException

Grading Scheme: -1 for each error to a maximum of 3 points each per part.

2. (12 points) Each of the following code snippets fails to compile in Java (thankfully!) because the last line would cause a runtime error. For each snippet below, **circle** the first line that causes a compile-time error and **briefly describe** the compilation error.

For example, if the snippet were:

```
boolean x = true;
int y = x;
int z = y + 3;
```

You would circle the second line because you cannot initialize an `int` variable with a `boolean` value, as `boolean` is not a subtype of `int`.

Below, assume that `List` and `ArrayList` been imported from `java.util`. For reference, an abbreviated version of the `List` interface appears in the companion handout.

- (a)

```
List<Point> s = new ArrayList<Point>();
List<Object> p = s;
p.add(new Object());
Point x = s.get(0);
double z = x.getX();
```

Answer: 2, List<Point> is not a subtype of List<Object>

- (b)

```
List<Object> s = new ArrayList<Object>();
List<Object> p = s;
p.add(new Object());
Point x = s.get(0);
double z = x.getX();
```

Answer: 4, the result of s.get is Object, not Point and Point is not a subtype of Object.

- (c)

```
List<Point> s = new ArrayList<Point>();
List<?> p = s;
p.add(new Object());
Point x = s.get(0);
double z = x.getX();
```

Answer: 3, as Object is not a subtype of ?

- (d)

```
List<? extends Displaceable> s = new ArrayList<Point>();
List<? extends Object> p = s;
p.add(new Object());
Displaceable x = s.get(0);
double z = x.getX();
```

Answer: 3, as Object is not a subtype of ? extends Object

Grading Scheme: 1 pt per circle, 2 pt per reason

3. (8 points) These multiple-choice questions test your knowledge of Java collections. All the interactions assume that the appropriate `import` statements have been used to make `List`, `ArrayList`, `Set`, and `HashSet` available. Abbreviated versions of the `Set` and `List` interface documentation appears in the companion hand-out. **Circle** the correct outcome from the interactions:

(a)

```
Set<String> s = new HashSet<String>();
s.add("a");
s.add("b");
s.add("b");
s.add("c");
s.size()
```

3 4 5 exception

Answer: 3

(b)

```
Set<String> s = new HashSet<String>();
s.add("a");
s.add("b");
s.add("b");
s.remove("b");
s.remove("b")
```

true false "b" exception

Answer: false

(c)

```
List<String> l = new ArrayList<String>();
l.add("a");
l.add("b");
l.add("c");
l.get(1)
```

"b" true false exception

Answer: "b"

(d)

```
List<String> l = new ArrayList<String>();
l.add("a");
l.add("b");
l.add("a");
l.remove("a");
l.remove("a");
l.get(0)
```

"a" "b" null exception

Answer: "b"

Grading Scheme: 2 pts per answer.

4. (18 points) Recall the `PointArrayList` class that we discussed in lecture. In this question, you will need to extend this class with additional methods. For reference, the complete definitions of `PointArrayList`, `PointList` and `Point` appear in the companion handout.

- (a) Implement the method `public void clear()` which removes all elements from the list. After a call to this method, the size of the list should be zero, and all calls to `contains` or `remove` should return `false`.

Answer:

```
public void clear() {
    this.size = 0;
}
```

Grading Scheme: 2 pts total. No partial credit.

- (b) Implement the method `public boolean allValid()` which returns `true` if all points in the list are not null and have non-negative x and y coordinates.

Answer:

```
public boolean allValid() {
    for (int i=0; i<size; i++) {
        Point p = data[i];
        if (p == null || p.getX() < 0 || p.getY() < 0)
            return false;
    }
    return true;
}
```

Grading Scheme: 4pts total. -2 for data.length instead of size. -2 for not checking p for null. -2 for wrong logic in test. Other errors at discretion.

(c) Implement the method

```
public int findFurthest()
```

which returns the index of the point that is the furthest from the origin. In your solution, you may use the following method in the `Point` class (which returns the square of the distance to that point):

```
public double getDistanceSq() {  
    return (x * x) + (y * y);  
}
```

If the point list does not contain any non-null points, the index -1 should be returned.

Answer:

```
public int findFurthest() {  
    double maxdist = -1;  
    int where = -1;  
    for (int i=0; i < size; i++) {  
        Point p = data[i];  
        if (p != null) {  
            double pdist = getDistanceSq(p);  
            if (pdist > maxdist) {  
                maxdist = pdist;  
                where = i;  
            }  
        }  
    }  
    return where;  
}
```

Grading Scheme: 6pts total. -2 for possible NullPointerException. -1 for not updating maxdistance. -1 for not updating where. -1 for never returning -1. -1 for not returning index. -1 for using length instead of size in loop conditions. Other errors at discretion.

(d) Implement the method

```
public PointList append(PointList l)
```

which appends the specified list to this point list.

The Points of the argument are appended to the end of this list changing its contents. This method increases the size of this list by the size of the argument. If `l` is `null`, then this list is unchanged. The method returns a reference to this object.

Answer:

```
public PointList append(PointList l) {
    if (l != null) {
        int newSize = this.size + l.size();
        if (newSize >= this.data.length) {
            grow(newSize);
        }
        for (int i=0; i < l.size(); i++) {
            this.data[i+size] = l.get(i);
        }
    }
    this.size = newSize;
    return this;
}
```

or: less efficient, but still correct.

```
public PointList append(PointList l) {
    if (l != null) {
        for (int i=0; i < l.size(); i++) {
            this.add( l.get(i) );
        }
    }
    return this;
}
```

Grading Scheme: 6 points total. -2 for not growing the list. -1 for not growing it enough. -2 (each) for getting indices to `this.data` and argument to `l.get` wrong. -2 for using `l.data` instead of `l.get` (Or `l.size` instead of `l.size()`). -2 for not checking if `l` is `null`. -2 for not having a loop to copy the data over. -2 for not updating `this.size`. -1 for wrong termination condition on loop. -2 for not returning `this`. Other errors at discretion.

List.java

```
1  /* A partial copy of Collections interfaces & classes from the java.util package
2   * so that we can see how it works. */
3
4  interface List<E> {
5      /** An ordered collection (also known as a sequence). The user of this interface has
6       precise control over where in the list each element is inserted. The user can
7       access elements by their integer index (position in the list), and search for
8       elements in the list.
9
10      Unlike sets, lists typically allow duplicate elements. More formally, lists
11      typically allow pairs of elements e1 and e2 such that e1.equals(e2), and they
12      typically allow multiple null elements if they allow null elements at all.
13      */
14
15      /** Add p to the end of the list. Always returns true. */
16      public boolean add(E p);
17
18      /** Remove the first occurrence of p, if present.
19       Returns true if this list contained the specified element. */
20      public boolean remove(Object p);
21
22      /** Test whether the list contains p */
23      public boolean contains(Object p);
24
25      /** Number of elements in list */
26      public int size();
27
28      /** Get the ith element of the list (starts from 0)
29       requires 0 <= i && i < size()
30       throws IndexOutOfBoundsException if i is out of range. */
31      public E get(int i);
32
33      /** Set the ith element to p
34       requires 0 <= i && i < size()
35       throws IndexOutOfBoundsException if i is out of range. */
36      public void set(int i, E p);
37
38 }
```

Set.java

```
1  /* A partial copy of Collections interfaces & classes from the java.util package
2   * so that we can see how it works. */
3
4  interface Set<E> {
5      /**
6       * A collection that contains no duplicate elements. More formally, sets contain no pair
7       * of elements e1 and e2 such that e1.equals(e2), and at most one null element. As implied
8       * by its name, this interface models the mathematical set abstraction.
9       */
10
11     /** Adds the specified element to this set if it is not already present.
12      * Returns true if this set did not already contain the specified element. */
13     public boolean add(E o);
14
15     /** Returns true if this set contains the specified element. */
16     public boolean contains(Object o);
17
18     /** Returns true if this set contains no elements. */
19     public boolean isEmpty();
20
21     /** Removes the specified element from this set if it is present.
22      * Returns true if the set contained the specified element (or equivalently, the set changed
23      * as a result of the call. The set will not contain the specified element once the
24      * call returns. */
25     public boolean remove(Object o);
26
27     /** Returns the number of elements in this set (its cardinality). */
28     public int size();
29
30 }
```

Point.java

```
1 public class Point implements Displaceable {
2     private double x,y;
3     public double getX() { return x; }
4     public double getY() { return y; }
5     public void move(double dx, double dy) {
6         x += dx;
7         y += dy;
8     }
9
10    public Point(double x, double y) {
11        this.x = x; this.y = y;
12    }
13 }
```

PointList.java

```
1  /* A partial copy of Collections interfaces & classes in java.util package
2   * so that we can see how it works. */
3
4  interface PointList {
5      /** Add p to the end of the list. Always returns true. */
6      public boolean add(Point p);
7
8      /** Remove the first occurrence of p, if present. */
9      public boolean remove(Object p);
10
11     /** Test whether the list contains p */
12     public boolean contains(Object p);
13
14     /** Number of elements in list */
15     public int size();
16
17     /** Get the ith element of the list (starts from 0)
18         requires 0 <= i && i < size() */
19     public Point get(int i);
20
21     /** Set the ith element to p
22         requires 0 <= i && i < size() */
23     public void set(int i, Point p);
24
25 }
```

PointArrayList.java

```
1  class PointArrayList implements PointList {
2      protected Point[] data;
3      protected int size;
4
5      public PointArrayList() {
6          this.data = new Point[10];
7          this.size = 0;
8      }
9
10     /** increase the length of the data array by at least 50% */
11     protected void grow(int newSize) {
12         if (newSize < (3*size)/2 + 1)
13             newSize = (3*size)/2 + 1;
14         Point[] newData = new Point[newSize];
15         for (int i = 0; i < data.length; i++)
16             newData[i] = data[i];
17         data = newData;
18     }
19
20     /** Add p to the end of the list. Always returns true. */
21     public boolean add(Point p){
22         if (size >= data.length)
23             grow(size+1);
24         data[size] = p;
25         size++;
26         return true;
27     }
28
29
30     /** Remove the first occurrence of p, if present. */
31     public boolean remove(Object p) {
32         int where = -1;
33         for (int i=0; i<size; i++) {
34             // if (data[i] == p)         checks reference equality
35             // if (data[i].equals(p))    doesn't allow data[i] to be null
36             // if (p.equals(data[i]))    doesn't allow p to be null
37             if (Util.equals(data[i],p)) {
38                 where = i;
39                 break;
40             }
41         }
42         if (where == -1) return false;
43         Point[] newData = new Point[data.length]; // could change length here.
44         for (int i=0; i<size-1; i++) {
45             if (i < where) {
46                 newData[i] = data[i];
47             } else {
48                 newData[i] = data[i+1];
49             }
50         }
51         data = newData;
52         size--;
53         return true;
54     }
55
56     /** Test whether the list contains p */
```

```

57     public boolean contains(Object p) {
58         for (int i=0; i<size; i++) {
59             if (Util.equals(data[i],p))
60                 return true;
61         }
62         return false;
63     }
64
65     /** Number of elements in list */
66     public int size() { return size; }
67
68     /** Get the ith element of the list (starts from 0)
69         requires 0 <= i && i < size() */
70     public Point get(int i) {
71         if (i < size)
72             return data[i];
73         else
74             throw new IndexOutOfBoundsException();
75     }
76
77
78     /** Set the ith element to p
79         requires 0 <= i && i < size() */
80     public void set(int i, Point p) {
81         if (i < size)
82             data[i] = p;
83         else
84             throw new IndexOutOfBoundsException();
85     }
86 }

```

Util.java

```

1  class Util {
2      /** Returns whether o1 equals o2 even if either
3       * reference could be null */
4      static boolean equals(Object o1, Object o2) {
5          if (o1 == null)
6              return o2 == null;
7          else
8              return o1.equals(o2);
9      }
10 }

```