

CSE 120

Midterm I — February 7, 2007

Name: _____

Student ID: _____
(from your PennCard)

Email _____@seas.upenn.edu

Lab:(circle one)

201	202	203	204
W 1:00	W 2:30	R 12:00	R 1:30

- 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 given as 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		13
2		12
3		13
4		12
Total		50

1. The first question considers the classes in files **ThreeInts.java** and **Boing.java** in the code handout.

(a) (3 points) What is the result of the following DrJava interaction:

```
> import threeints.*;
> ThreeInts ti = new ThreeInts(1,2,3);
> Boing b1 = new Boing(1,ti);
> Boing b2 = new Boing(2,ti);
> b1.nosh()
{ 2 3 4 }
> b2
```

Answer: { 2 3 4 }

(b) (3 points) Suppose we define the class **NewThreeInts** that overrides the definition of **incr** in the class **ThreeInts**.

```
public class NewThreeInts extends ThreeInts {
    protected ThreeInts incr() {
        return new NewThreeInts(data[0]+1, data[1]+1, data[2]+1);
    }
    public NewThreeInts(int i1, int i2, int i3) {
        super(i1,i2,i3);
    }
}
```

What is the result after the analogous interaction?

```
> import threeints.*;
> ThreeInts tj = new NewThreeInts(1,2,3);
> Boing b3 = new Boing(1,tj);
> Boing b4 = new Boing(2,tj);
> b3.nosh()
{ 2 3 4 }
> b4
```

Answer: { 1 2 3 }

- (c) (7 points) Given the definition of `NewThreeInts` in question(b) write “true”, “false”, or “error” in each box below to indicate the DrJava output.

```
> import threeints.*;
> ThreeInts t1 = new NewThreeInts(1,2,3);
> ThreeInts t0;
> t1 == t0
```

Answer: false

```
> t1 instanceof NewThreeInts
```

Answer: true

```
> t0 instanceof ThreeInts
```

Answer: false

```
> t1.incr() == t0.incr()
```

Answer: error

```
> t0 = new NewThreeInts(1,2,3);
> t1 == t0
```

Answer: false

```
> t0 = t1;
> t1 == t0
```

Answer: true

```
> t1.incr() == t0.incr()
```

Answer: false

2. There are three *compilation* errors in the following code fragment. Circle each error, mark it with a letter (a)-(c), and in the space below, *briefly* describe what caused the error. There is only one error per line. (Don't waste time trying to figure out what this program does!)

```
interface Plugh {
    public String xyzzzy ();
}

abstract class Frobozz implements Plugh {

    private int count = 69105;

    public abstract int getCount();

    public String xyzzzy () { return "Nothing happens."; }

    public void foozle() {
        Plugh x = this;
        if (x.getCount() >= 10)
            return;
        incr();
    }

    public Frobozz issue() { return new Frobozz(); }

    public static void incr() { this.count++; }
}
```

- (a) (4 points) *Answer:* The local variable `x` in the method `foozle` cannot call the method `getCount` because its static type is `Plugh`.
- (b) (4 points) *Answer:* `Frobozz` is an abstract class, it cannot be instantiated in the method `issue`.
- (c) (4 points) *Answer:* Cannot access dynamic instance variable `count` in static method `incr`.

Grading scheme:

- -3 if the right location is identified but the wrong reason is given.
- 2/4 for identifying `return` or a lack of `return` in `foozle` as an error. It's not an error, but we didn't cover this adequately in class.
- 2/4 for identifying the call to `incr` in `foozle` as an error. It's not an error, but we didn't cover this adequately in class.
- -1 for saying in part (b) that the class cannot be instantiated because it does not have a constructor, without noting that abstract classes can never be instantiated, even if they do have constructors.
- Other errors at discretion.

3. This problem concerns the two Java source files, **TicketFactory.java** and **Ticket.java**, found in the code handout. Please take a minute to look over these source files.

Your job is to implement the class **OnePerFactory** that adapts the **TicketFactory** class so that a single person can buy at most *one* ticket from a *particular* ticket factory.

A sample interaction with this new class might be as follows:

```
> import ticket.*;
> OnePerFactory tf = new OnePerFactory("King Tut", 200);
> Ticket t0 = tf.issue("Alice");
> t0
Ticket # 0 issued to Alice
> Ticket t1 = tf.issue("Bob");
> t1
Ticket # 1 issued to Bob
> Ticket t2 = tf.issue("Alice");
> t2
null
```

The beginning of the definition of class **OnePerFactory** is below. This definition includes the declaration of two instance variables: **tickets** should store an array of tickets issued by the factory, and **ticketFactory** is an instance of the **TicketFactory** class. Your task will be to implement the constructor and two methods for this class. You may not add any new instance variables, nor change the declarations below.

```
package ticket;

public class OnePerFactory {
    // Array of tickets that have been issued by this factory.
    private Ticket[] tickets;

    // A ticket factory
    private TicketFactory ticketFactory;
```

- (a) (3 points) Complete the constructor for this class.

```
    public OnePerFactory(String name, int maxTickets) {

        tickets = new Ticket[maxTickets];
        ticketFactory = new TicketFactory(name, maxTickets);

    }
```

Grading scheme: -1 if tickets is not initialized. -2 if ticketFactory is not initialized. Other errors at discretion.

- (b) (4 points) Implement a helper method of the class `OnePerFactory` that determines whether a person has already bought a ticket.

```
private boolean bought(String purchaser) {  
  
    for (int i=0; i<ticketFactory.getIssued(); i++) {  
        if (tickets[i].getPurchaser().equals(name)) {  
            return true;  
        }  
    }  
    return false;  
  
}
```

Grading scheme: -1 if the method throws a `NullPointerException` when accessing `getPurchaser`. -1 for accessing private members of class `TicketFactory`. No points deducted for using `=` or `==` instead of `equals`. Other errors at discretion.

- (c) (6 points) Implement the `issue` method to make sure that each purchaser can only buy one ticket. If a purchaser has previously bought a ticket, this method should return `null`.

```
public Ticket issue(String purchaser) {  
  
    if (bought(purchaser))  
        return null;  
    else {  
        Ticket t = ticketFactory.issue(purchaser);  
        if (t == null) {  
            return null;  
        }  
        tickets[ticketFactory.getIssued()-1] = t;  
        return t;  
    }  
  
}
```

Grading scheme: -1 if the method throws a runtime exception. -1 for accessing private members of class `TicketFactory`. -1 for not returning an answer. -1 for not using `ticketFactory.issue`. -2 for not storing into the `tickets` array. Other errors at discretion.

4. Sudoku is a logic puzzle where players are given a partially filled 9x9 grid and the instructions: "Fill in the grid so that every row, every column, and every 3x3 box contains the digits 1 through 9." For example, a solved grid that satisfies these constraints is below.

9	6	3	1	7	4	2	5	8
1	7	8	3	2	5	6	4	9
2	5	4	6	8	9	7	3	1
8	2	1	4	3	7	5	9	6
4	9	6	8	5	2	3	1	7
7	3	5	9	6	1	8	2	4
5	8	9	7	1	3	4	6	2
3	1	7	2	4	6	9	8	5
6	4	2	5	9	8	1	7	3

This problem concerns two methods from a Sudoku solution verifier.

- (a) (5 points) The following method checks that the numbers stored in the 2-D array `grid` satisfy the first part of the constraint: that every row contains the digits 1 through 9. Unfortunately, this method does no error checking of its input.

```

1 void boolean checkRows(int[][] grid) {
2     for (int i = 0; i < 9 ; i++) {
3         boolean[] used = new boolean[9];
4         for (int j = 0; j < 9; j++) {
5             int here = grid[i][j]-1;
6             if (used[here])
7                 return false;
8             used[here] = true;
9         }
10    }
11    return true;
12 }
```

Which line or lines in this method could potentially raise an `ArrayIndexOutOfBoundsException`? Briefly explain why?

Answer: 6 and 7. For the first case, the array `grid` may not be exactly 9x9. If it is any smaller an exception will occur.

In the second case, the numbers stored in `grid` may not be between 1-9 (inclusive).

Line 8 cannot cause an exception, because it would have been triggered on line 7.

Grading scheme: -1 point each for not identifying the line. -1 point each for not giving a proper explanation. -1 point each for identifying a false location.

(b) (7 points)

The next method should determine if each 3x3 box uses the numbers 1 through 9 exactly once. This method iterates through the nine 3x3 boxes, and then within each of those boxes, iterates through the nine cells.

Fill in the blanks to complete this method.

```
void boolean checkBoxes(int [][] grid) {

    for (int x = ____0_____; x < ____3_____; x++) {

        for (int y = ____0_____; y < ____3_____; y++) {

            boolean[] used = new boolean[9];

            for (int i = ____3 * x_____; i < ____(3 * x) + 3_____; i++) {

                for (int j = __3 * y _____; j < ___(3 * y) + 3_____; j++) {

                    int here = grid[i][j]-1;

                    if (used[here])

                        return false;

                    used[here] = true;
                }

            }

        }

    }

    return true;

}
```

TicketFactory.java

```
1  package ticket;
2
3  public class TicketFactory {
4      private String name;        // name of event
5      private int issued;        // number of tickets that have been issued
6      private int maxTickets;    // maximum number of tickets available
7
8      // constructor, initializes instance variables
9      public TicketFactory(String name, int maxTickets) {
10         this.name = name;
11         this.maxTickets = maxTickets;
12         this.issued = 0;
13     }
14
15     // accessor methods for instance variables
16     public int getIssued()      { return issued; }
17     public int getMaxTickets() { return maxTickets; }
18     public String getName()    { return name; }
19
20     // Factory method: issue a ticket to a particular
21     // purchaser as long as the maximum number of tickets
22     // has not been reached. Return null otherwise.
23     public Ticket issue(String purchaser) {
24         if (issued < maxTickets) {
25             Ticket t = new Ticket(purchaser, issued);
26             issued++;
27             return t;
28         }
29         else {
30             return null;
31         }
32     }
33
34
35 }
36
```

Ticket.java

```
1  package ticket;
2
3  public class Ticket {
4      private String purchaser; // the purchaser of this ticket
5      private int number;      // the ticket number
6
7      // Note: Constructor has package scope
8      Ticket(String purchaser, int number) {
9          this.purchaser = purchaser;
10         this.number = number;
11     }
12
13     // accessor methods for instance variables.
14     public String getPurchaser() { return purchaser; }
15     public int    getNumber() { return number; }
16
17     public String toString() {
18         return "Ticket # " + number + " issued to " + purchaser;
19     }
20 }
```

ThreeInts.java

```
1  package threeints;
2
3  public class ThreeInts {
4      protected int[] data;
5      public ThreeInts(int i1, int i2, int i3) {
6          data = new int[3];
7          data[0] = i1; data[1] = i2; data[2] = i3;
8      }
9      protected ThreeInts incr() {
10         for (int i=0; i<data.length; i++) { data[i]++; }
11         return this;
12     }
13     public String toString() {
14         return "{ " + data[0] + " " + data[1] + " " + data[2] + " }";
15     }
16
17 }
```

Boing.java

```
1  package threeints;
2
3  public class Boing {
4      private int number;
5      private ThreeInts data;
6
7      public Boing(int number, ThreeInts data) {
8          this.number = number;
9          this.data = data;
10     }
11
12     public Boing nosh() {
13         for (int i = 0; i < number; i++)
14             data = data.incr();
15         return this;
16     }
17
18     public String toString() {
19         return data.toString();
20     }
21 }
```