



EXAMINATIONS — 2012

MID-YEAR

**COMP 102
INTRODUCTION TO
COMPUTER PROGRAM
DESIGN**

Time Allowed: 3 Hours

Instructions: Attempt ALL Questions.

Answer in the appropriate boxes if possible — if you write your answer elsewhere, make it clear where your answer can be found.

The exam will be marked out of 180 marks.

Only silent non-programmable calculators or silent programmable calculators with their memories cleared are permitted.

Non-electronic foreign language dictionaries are permitted.

Java Documentation will be provided with the exam script

There are spare pages for your working and your answers in this exam, but you may ask for additional paper if you need it.

Questions

	Marks
1. Understanding Java	[57]
2. Files	[27]
3. Arrays of Objects	[26]
4. Interface classes	[12]
5. Event driven input	[15]
6. 2D Arrays	[23]
7. Debugging loops	[20]

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 1. Understanding Java

[57 marks]

(a) [4 marks] What will the following fragment of Java print out?

```

double x = 1;
double y = x;
while (y < 16){
    y = y * 2;
    UI.println (x + " : " + y);
    x = x + 1;
}
UI.println ("done");

```

x: y:

(b) [5 marks] Almost every line of the following **bad** method has a syntax error that the compiler would complain about. Circle, and briefly explain each error

```

public String bad(int x; int y){

    while (x != y){

        size = size + y;

        x = x + 1;

    }

    else {

        int y = 16;

        return (x + y);

    }

}

```

(Question 1 continued on next page)

(Question 1 continued)

(c) [6 marks] The conditionalTest method below has one parameter and prints out a message about the parameter.

```
public void conditionalTest( int datum){
    String ans = "";
    if ( datum > 3 && datum < 10) {
        ans = "first";
    }
    else if ( datum < 5 || datum > 100) {
        ans = "last";
    }
    else {
        ans = "all";
    }

    ans = ans + " " + datum;

    if (datum % 2 == 0) {
        Ul.println (ans + " done");
    }
    else {
        Ul.println (ans + " quit");
    }
}
```

What would the following calls to conditionalTest print out?

Hint: Note that % is the remainder operator.

conditionalTest(4); \Rightarrow

conditionalTest(12); \Rightarrow

conditionalTest(1); \Rightarrow

(Question 1 continued on next page)

(Question 1 continued)

(d) [5 marks] Complete the following `triangleArea` method so that it computes and returns the area of a triangle. It should have two parameters (doubles) which are the width and the height of a triangle.

Note: the area of a triangle is half the product of the width and the height.

```
public      triangleArea
}

```

(e) [5 marks] Consider the following `reduce` method.

```
public void reduce (int first , int last){
    UI.println ("reduce (" + first + " , " + last + " )");
    while (last > 0){
        if ( first <= last){
            last = last - first ;
        }
        else {
            int temp = last;
            last = first - last;
            first = temp;
        }
        UI.println ( first + " : " + last);
    }
}

```

first: last: temp:

What will be printed in the text pane if `reduce` is called with the arguments 6 and 16? (The first line is done for you.)

```
reduce (6, 16)
```

(Question 1 continued on next page)

(Question 1 continued)

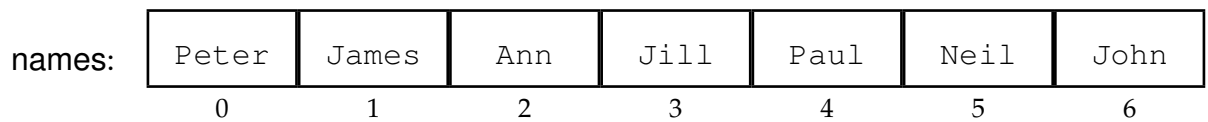
(f) [7 marks] Consider the following mixNames method which modifies an array of Strings.

```
public void mixNames(String [ ] data){  
    int index=0;  
    for( int i=0; i<data.length; i++){  
        if (data[i]. startsWith("J")){  
            data[index] = data[i];  
            data[i] = null;  
            index++;  
        }  
    }  
}
```



Suppose that the variable names is defined as follows:

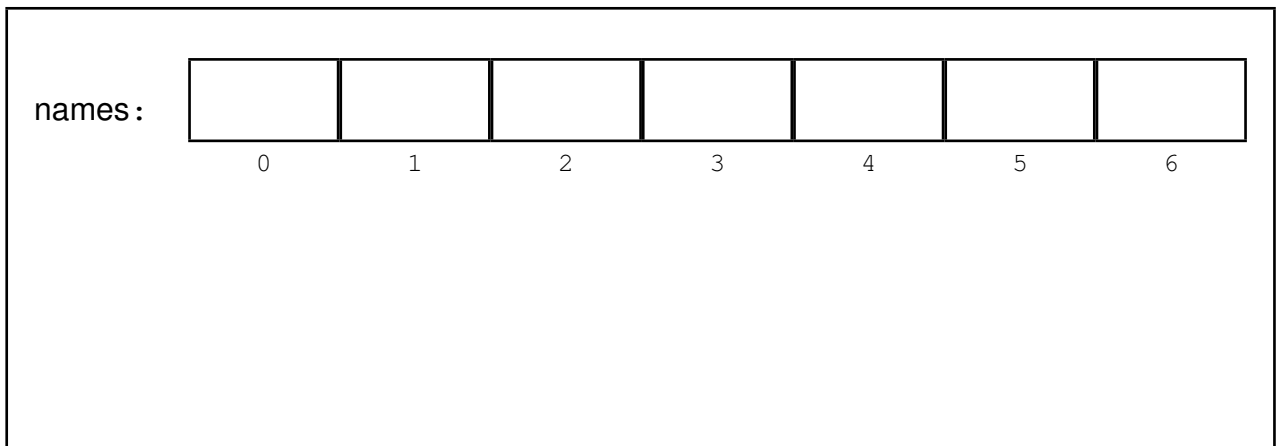
```
String [ ] names = new String[ ]{"Peter","James","Ann","Jill","Paul","Neil","John"};
```



Show the contents of names after the method call

```
mixNames(names);
```

Hint: show your working.



(Question 1 continued on next page)

(Question 1 continued)

(g) [7 marks] Suppose the file `games.txt` contains the text:

```
Blues Yellows 15 4
Greens Blacks 8 9
Purples Reds 7 3
```

What will the following `printGames` method print out?

```
public void printGames(){
    try{
        Scanner scan = new Scanner(new File("games.txt"));
        int teams = 0;
        scan.next();
        while ( scan.hasNext() ){
            teams++;
            if (scan.hasNextInt()){
                int score = scan.nextInt() - scan.nextInt();
                UI.println (" by " + score);
            }
            else {
                UI.println ("Team: " + scan.next());
            }
        }
        scan.close();
        UI.println ("Teams: " + teams);
    }
    catch(IOException e){UI.println("File reading failed");}
}
```

teams:

(Question 1 continued on next page)

(Question 1 continued)

(h) [8 marks] The `AssemblyJob` class below defines `AssemblyJob` objects, which have three fields to store an item category, a worker's name, and the time spent on the job. The class defines a constructor and three methods.

```
public class AssemblyJob{
    private String category;
    private String worker;
    private int mins;

    public AssemblyJob(String what, String who){
        this.category = what;
        this.worker = who;
    }

    public void print(){
        UI.printf ("%s on %s for %d mins\n",
            this.worker, this.category, this.mins);
    }

    public void workOn (int m){
        this.mins = this.mins + m;
    }

    public void takeOver(AssemblyJob component){
        this.mins = this.mins + component.mins;
        component.mins = 0;
        if ( ! this.worker.equals(component.worker) ) {
            component.worker = this.worker;
        }
    }
}
```

category:	<input type="text"/>
worker:	<input type="text"/>
mins:	<input type="text"/>

category:	<input type="text"/>
worker:	<input type="text"/>
mins:	<input type="text"/>

category:	<input type="text"/>
worker:	<input type="text"/>
mins:	<input type="text"/>

(Question 1 continued on next page)

(Question 1 continued)

What will the following testJobs method print out?

```
public static void testJobs(){  
  
    AssemblyJob jobD = new AssemblyJob("Drawer", "Bill");  
  
    jobD.workOn(20);  
    Ul.print ("1: "); jobD.print ();  
  
    AssemblyJob jobC = new AssemblyJob("Cabinet", "James");  
  
    jobC.workOn(30);  
    jobD.workOn(10);  
    Ul.print ("2: "); jobC.print ();  
    Ul.print ("3: "); jobD.print ();  
  
    jobC.takeOver(jobD);  
    Ul.print ("4: "); jobC.print ();  
    Ul.print ("5: "); jobD.print ();  
  
    AssemblyJob jobM = new AssemblyJob("Mirror", "Bill");  
  
    jobM.workOn(50);  
    jobD.workOn(15);  
    jobC.workOn(40);  
    Ul.print ("6: "); jobC.print ();  
    Ul.print ("7: "); jobD.print ();  
    Ul.print ("8: "); jobM.print ();  
}
```

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

(Question 1 continued on next page)

(Question 1 continued)

(i) [10 marks] Complete the `RentalTool` class on the facing page which stores information about tools at a renting agency.

A `RentalTool` object should contain three fields:

- `toolType`, which contains the type of tool (e.g. "Power Drill")
- `purchased`, which contains the year that the tool was purchased (e.g. 2009)
- `rentalDays`, which contains the total number of days that the tool has been rented out (initially 0).

`RentalTool` should have a constructor that takes one `String` parameter and one integer parameter and stores them in the `toolType` and `purchased` fields.

`RentalTool` should have three methods:

- `getDays`, which returns the number of days the tool has been rented out.
- `rent`, which is called when the tool is rented out, and is passed the number of days in the rental period. It should update the `rentalDays` field.
- `toString()`, which returns a `String` containing the name of the tool, the year purchased, and the number of rental days in the format
`"Power Drill (2009) 391 days"`

(Question 1 continued on next page)

(Question 1 continued)

```
public class RentalTool{
```

```
    // fields
```

```
    // constructor
```

```
    // methods
```

```
}
```

Question 2. Files

[27 marks]

Suppose the file `biscuits.txt` contains data about packets of biscuits in a store. Each line of the file contains information about one kind of biscuit - the category, the price for the packet, the name of the biscuit, and the packet size (number of biscuits in each packet). For example, the following might be the first nine lines of the file.

chocolate	3.40	tim tam	17
plain	1.90	ginger nuts	20
chocolate	2.80	mallowpuffs	10
plain	2.20	macaroons	15
filled	2.45	creme shortbread	23
chocolate	3.80	fudge brownie	12
plain	2.10	oat raisin bar	14
filled	2.80	shrewsbury	18
plain	2.00	anzac biscuits	21

The category is always a single word, but the biscuit name may consist of multiple words.

(a) [7 marks] Complete the following `listBudget` method, whose parameter is a maximum price. `listBudget` should read the file and print out the name and packet size of each biscuit whose price is not above the specified maximum.

For example, on the data above, `listBudget(2.00);` should print

```
Biscuits up to $2.00:
    ginger nuts          20
    anzac biscuits      21
```

```
public void listBudget (double maxPrice){
    String filename = "biscuits.txt";
    UI.printf ("Biscuits up to $%4.2f:\n", maxPrice);
    try{
        Scanner sc = new Scanner (new File (filename));

        sc.close ();
    } catch (IOException e){UI.println("fail");}
}
```

(Question 2 continued)

(b) [10 marks] Complete the following `minPrice` method which should print out the biscuit (and packet size) that is the cheapest of a given category. You may assume there is at least one biscuit in the category.

For example, if the file had only the nine lines in the example above, `minPrice("chocolate")` should print out

```
Lowest price chocolate biscuit is
  mallowpuffs          10
```

```
public void minPrice(String biscuitType){
    String filename = "biscuits.txt";
    UI.println ("Lowest price " + biscuitType + " biscuit is");
    try{
        Scanner sc = new Scanner (new File (filename));

        sc.close ();
    }
    catch(IOException e){UI.println("Fail: " + e);}
}
```

(Question 2 continued on next page)

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 2 continued)

(c) [10 marks] Complete the following `pricePerBiscuit` method which prints out the biscuit name and price for an individual biscuit for each line in the file. (Note that it must use the number of biscuits in the packet to compute this.)

For example, the output for the first four lines of the `biscuits.txt` file above should be:

```
Prices per individual biscuit:
$0.20 each for tim tam
$0.10 each for ginger nuts
$0.28 each for mallowpuffs
$0.15 each for macaroons
```

```
public void pricePerBiscuit(){
    String filename = "biscuits.txt";
    UI.println("Prices per individual biscuit:");
    try{
        Scanner sc = new Scanner(new File(filename));

        sc.close();
    }
    catch(IOException e){UI.println("Fail: " + e);}
}
```

Question 3. Arrays of Objects

[26 marks]

This question concerns a FlatFinder program to let users examine a list of available flats. The FlatFinder class (first ten lines shown on the facing page) stores the information about the flats in a field containing an array of Flat objects. It also has a count field that contains the number of Flat objects, and the Flat objects are stored in cells 0 through count-1 of the array.

The Flat class, for representing information about individual flats, is shown below. You should use the appropriate Flat methods in your answers.

```
public class Flat{
    // fields
    private String address;
    private String landlord;
    private int bedrooms; // number of bedrooms
    private double rent; // weekly rent
    // constructor
    public Flat(String addr, String landl, int beds, double rnt){
        this.address = addr;
        this.landlord = landl;
        this.bedrooms = beds;
        this.rent = rnt;
    }
    //methods
    public void printDetails (){
        UI.printf ("%d beds, $%.0f /week, %s (%s)\n",
            this.bedrooms, this.rent, this.address, this.landlord );
    }
    public String getAddress() {
        return this.address;
    }
    public boolean hasLandlord(String name){
        return (this.landlord.equals(name));
    }
    public int getBeds() {
        return this.bedrooms;
    }
    public double getRent() {
        return this.rent;
    }
}
```

(Question 3 continued)

The following are some of the fields and one of the methods of the FlatFinder class:

```
public class FlatFinder{

    private Flat [ ] flats = new Flat [1000];
    private int count = 0;

    public void listFlats (){
        for ( int i=0; i<this.count; i++){
            Ul.print (i+": ");
            this.flats [ i ]. printDetails ();
        }
    }
}
```

(a) [6 marks] Complete the following addFlat method of the FlatFinder class. Its parameter is a Flat and it should add the given Flat to the flats array. If the array is full, the method should print out a message. (You do not need to check for duplicates.)

```
public void addFlat (Flat flat ){
```

```
}
```

(Question 3 continued on next page)

(Question 3 continued)

(b) [6 marks] Complete the following `findMostExpensive` method. It should return the address of the most expensive flat. If there is a tie for most expensive, it should return the earliest in the list.

```
public String findMostExpensive(){
```

```
}
```

(c) [6 marks] Complete the following `landlordStats` method which has one parameter — the name of a landlord — and should print both the number of flats of the given landlord, and their average rent. If the landlord has no flats, print “No Flats”.

```
public void landlordStats(String name){
```

```
}
```

(Question 3 continued on next page)

(Question 3 continued)

(d) [8 marks] Complete the following `removeListing` method in the `FlatFinder` class. Its parameter is *String* specifying an address, and it should remove the flat at that address from the list of flats and return `true`. If there is no flat with that address, it should return `false`. When removing the flat, it should not change the order of the other flats in the array. It should not introduce any nulls in positions 0 to `count-1`.

```
public boolean removeListing(String addr){
```

```
}
```

Question 4. Interface classes

[12 marks]

This question addresses the design of a computer adventure game in which the player has to collect items to solve a puzzle. Part of the program must store the items that the player has found so far. There are quite a number of different kinds of items: tools, maps, potions, food, jewels, etc. The different kinds of items behave differently, so there is a separate class for each kind of item. However, every item will have the following things in common:

- fields to store values such as the name of the item, the trade-in value of the item, and the weight of the item.
- an `infoString` method that returns a short `String` containing a description of the item.
- `getValue` and `getWeight` methods that return the trade-in value and the weight of the item.
- an `isWorking` method that returns a boolean value representing whether the item is working or broken.
- an `applySpell` method with one parameter (a `Spell` object) that applies the spell to the item (which may modify the item in some way).

The program must have a single array containing all the items that the player has currently collected. To do this, we need an interface class called `GameItem` that specifies a type that includes all the different kinds of items.

(a) [6 marks] Complete the following definition of the `GameItem` interface class.

```
public interface GameItem {
```

```
}
```

(Question 4 continued on next page)

(Question 4 continued)

(b) [6 marks] The following `Map` class describes map items, and provides several methods for maps. In the game, maps are very simple items that can be displayed to the user, and nothing else. They always work, and no spells affect them. Their description consists just of the `mapName`.

However, `Map` is not correctly written to make every `Map` object also be a `GameItem`. Modify the definition of `Map` below so that a `Map` correctly implements the `GameItem` interface class.

```
public class Map {
    private String mapName;
    private String pictureFile ;
    private double weight = 0.1;
    private double value;

    public Map(String mapNm, String fileName, double val){
        this.mapName = mapNm;
        this.pictureFile = fileName;
        this.value = val;
    }
    public double getValue(){return value;}

    public double getWeight(){return weight;}

    public void display(double x, double y){
        UI.drawImage(this.pictureFile, x, y);
    }
}
```

Question 5. Event Driven Input

[15 marks]

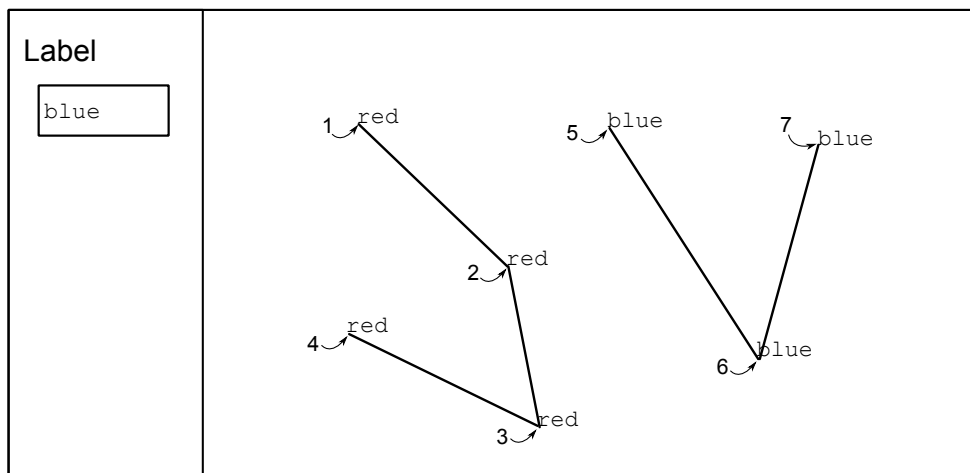
Complete the PolyLine program on the facing page so that it allows the user to draw labeled poly-lines (connected line segments) on the graphics pane. The program should have one textfield called "Label" and should also listen to the mouse.

Every time the user enters a value in the textfield, the program should remember the value in the field called lineLabel, and should also reset lastX and lastY to start a new polyline.

When the user clicks the mouse in the graphics pane (ie, the "clicked" action), the program should draw the current value of lineLabel at the point, and should also draw a line connecting the point to the previous point, unless it has just started a new polyline.

The diagram below shows what the program should do if the user

- entered the value "red" in the textfield,
- clicked the mouse at the positions labeled 1, 2, 3, and then 4
- entered the value "blue" in the textfield,
- clicked the mouse at the positions labeled 5, 6, 7



(Question 5 continued)

```
public class PolyLine implements UITextFieldListener, UIMouseListener{
    private double lastX = -1; // -1 indicates it is ready to start a new polyline.
    private double lastY = -1;
    private String lineLabel = "";

    public PolyLine(){

    }

    public void textFieldPerformed(String field , String value){

    }

    public void mousePerformed(String action, double x, double y) {

    }

}
```


(Question 6 continued)

(b) [7 marks] Complete the following `checkColumn` method whose parameter is the index of a column of the puzzle, and should return `true` if all the cells in the given column of a puzzle are the same color, and `false` otherwise.

For example, `checkColumn(3)` on the puzzle shown above should return `true`.

```
public boolean checkColumn(int col){
```

```
}
```

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 6 continued)

(c) [10 marks] Complete the following `shiftLeft` method which should shift all the colors in the puzzle one step to the left, except the colors in the leftmost column which should be moved to the rightmost column.

For example, given the puzzle above, `shiftLeft` should change it to:

R	b	P	B	B	B
R	W	P	G	b	G
R	W	P	R	b	G
W	W	P	W	b	G
B	W	P	R	B	b
R	b	P	B	G	G

```
public void shiftLeft () {
```

```
}
```

Question 7. Debugging Loops

[20 marks]

The following `mergeltems` method is intended to merge some numbers into a list of numbers, only adding numbers that are not already in this list. The list is represented by an array and a count, and `mergeltems` may assume that the array has enough extra space for all the new numbers.

For example, given a list of 5 numbers:

list:	3	4	7	6	4	0	0	0	0	0
	0	1	2	3	4	5	6	7	8	9
count:	5									
	0									

and an array of four new numbers to merge:

items:	5	4	2	5
	0	1	2	3

`mergeltems(list, 5, items)` should change the list to be

list:	3	4	7	6	4	5	2	0	0	0
	0	1	2	3	4	5	6	7	8	9

It should not add 4 since it is already in the list, and 5 is in the list when the second 5 in items is encountered.

This version of `mergeltems` has multiple errors.

```
1  /* merge items into list . Assumes list is large enough */
2  public void mergeltems (int [ ] list , int count, int [ ] items){
3      boolean found = false;
4      for (int i=0; i<items.length; i++){
5          for (int j=0; j<count; j++){
6              found = (items[i] == list [j]);
7          }
8          if (found) {
9              count++;
10             list [count] = items[i];
11         }
12     }
13 }
```

(Question 7 continued on next page)

(Question 7 continued)

(a) [6 marks] If this version of `mergeItems` (with the errors) were called with the arrays above:

```
mergeItems(list, 5, items);
```

what would the contents of the `list` array be when the method was finished?

Hint: show your working.

0	1	2	3	4	5	6	7	8	9	

(b) [6 marks] Identify and briefly describe three of the errors in the version of `mergeItems` above. You may answer by annotating the code on the facing page, or list line numbers and describe the errors in the box below.

(Question 7 continued on next page)

(Question 7 continued)

(c) [8 marks] Write a correct version of mergeItems.

```
/* merge items into list . Assumes list is large enough */  
public void mergeGood (int [ ] list , int count, int [ ] items){
```

```
}
```

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.