AP COMPUTER SCIENCE A – SEMESTER 2 FINAL EXAM STUDY GUIDE

1) What is the output of the code segment?

```
int [][] a=new int [5][];
/**/
int k=1, n=0;
for ( int i=0; i<5; ++i ) {
  a[i]=new int [k];
  for ( int j=0; j<k; ++j ) {
    a[i][j]=++n;
  }
  ++k;
}
/**/
String s;
for ( int i=0; i<a.length; ++i ) {
  s="";
  for ( int j=0; j<a[i].length; ++j ) {
    s += String.format("%3d",a[i][j]);
  }
  System.out.println(s);
}
```

Answer:

```
 1
 2  3
 4  5  6
 7  8  9 10
11 12 13 14 15
```

2) The class `Time.java`, listed below, gives the current Greenwich Mean Time (GMT) in 24-hour format.

```
import java.util.Calendar;
import java.util.TimeZone;
/*
 *  The time in Greenwich Mean Time
 */
public class Time {
  /**/
  private int hour, minute;
  /**/
  public Time() {
    /**/
    Calendar c=Calendar.getInstance(TimeZone.getTimeZone("GMT"));
    hour=c.get(Calendar.HOUR_OF_DAY);
    minute=c.get(Calendar.MINUTE);
  }
  /*
   *  hour = 0 - 23 (in Greenwich Mean Time)
   */
  public int getHour () {
    return hour;
  }
  /**/
  public int getMinute () {
    return minute;
  }
  /**/
  public String toString () {
    return String.format("%02d",getHour()) + ":" + String.format("%02d",getMinute());
  }
}
```

The class `CentralEuropeanTime.java`, listed below, extends `Time` to give the time, in 24-hour format, for Paris, France, which is one hour ahead of GMT.

```
 1: public class CentralEuropeanTime extends Time {
 2:   /**/
 3:   public CentralEuropeanTime () {
 4:     super();
 5:   }
 6:   /**/
 7:   @Override
 8:   public int getHour () {
 9:     /**/
10:     int h;
11:     /**/
12:     /* missing */
13:     /*  code   */
14:     return h;
15:   }
16: }
```

What is the required code on lines `12` and `13`?

Answer:

```
h=super.getHour()+1;
if ( h > 23 ) h -= 24;
```

This page intentionally is left blank.

3) We have a class `Sort.java`, listed here,

```
import java.util.ArrayList;
/**/
public class Sort<T> {
  /*
   *  Sort the ArrayList<T> into ascending order
   */
  public static <T extends Sortable> void sort ( ArrayList<T> alt ) {
    /*
     *  Implementation not shown
     */
  }
}
```

which will sort a list of objects of type `T` if class `T` implements the interface `Sortable.java`, listed here.

```
public interface Sortable {
  boolean lessThan ( Sortable s );
}
```

We want to use class `Sort` to sort a list of all the integers between `0` and `9999` (inclusive), ordered randomly, which list is contained in the file `unsorted_10000.txt`, the first few lines of which are shown here.

```
1055
 294
 593
3850
7680
 .
 .
 .
```

Consequently, we need a class `MyInteger.java` which implements `Sortable`. The class is listed at the top of the next page.

```
 1: public class MyInteger implements Sortable {
 2:   /**/
 3:   private int value;
 4:   /**/
 5:   public MyInteger ( int anInt ) {
 6:     value=anInt;
 7:   }
 8:   /**/
 9:   public int intValue () {
10:     return value;
11:   }
12:   /**/
13:   public String toString () {
14:     return String.format("%4d",value);
15:   }
16:   /*
17:    *  As per interface Sortable
18:    */
19:   public boolean lessThan ( Sortable s ) {
20:     /**/
21:     boolean rv;
22:     MyInteger otherInt;
23:     /**/
24:     otherInt=(MyInteger)s;
25:     /* missing */
26:     /*  code   */
27:     return rv;
28:   }
29: }
```

Finally, the class `DoSort.java`, listed below, performs the sort (see line `8`).

```
 1: import java.util.ArrayList;
 2: /**/
 3: public class DoSort {
 4:   /**/
 5:   public static void main ( String [] arg ) {
 6:     /**/
 7:     ArrayList<MyInteger> intList=readListFromFile("unsorted_10000.txt");
 8:     Sort.<MyInteger>sort(intList);
 9:     printListToFile("sorted_10000.txt",intList);
10:     /**/
11:     return;
12:   }
13:   /**/
14:   public static ArrayList<MyInteger> readListFromFile ( String fn ) {
15:     /*
16:      *  Implementation not shown
17:      */
18:   }
19:   /**/
20:   public static void printListToFile ( String fn, ArrayList<MyInteger> almi ) {
21:     /*
22:      *  Implementation not shown
23:      */
24:   }
25: }
```

Provide the two lines of code which should be on lines `25` and `26` of class `MyInteger`.

Answer:

either

```
if ( value < otherInt.intValue() ) rv=true;
else                               rv=false;
```

or

```
if ( otherInt.intValue() > value ) rv=true;
else                               rv=false;
```

4) Class `Vegetable` has a constructor which takes one `String`, and `Vegetable` implements the interface `FoodGroup`. Which of the following lines of code will compile with no errors?

```
1: FoodGroup fg=new Vegetable("Carrot");
2: Vegetable lettuce=new FoodGroup("Lettuce");
3: Vegetable pea=new Vegatable("Pea");
4: Object beanObject=new Vegatable("Bean");
```

Answer:

Lines `1`, `3` and `4`.

5) Consider the abstract class `Currency`

```
public abstract class Currency {
  /**/
  public Currency () { }
  /**/
  public abstract double getUSDvalue ();
  /**/
  public String toString() {
    return "$" + String.format("%4.2f",getUSDvalue());
  }
}
```

and the (concrete) class `Euro` which extends `Currency`.

```
public class Euro extends Currency {
  public Euro ( ) { super(); }
  public double getUSDvalue() { return 1.09; }
}
```

What is the output generated by running `RunCurrency.java` as listed?

```
public class RunCurrency {
  /**/
  public static void main ( String [] arg ) {
    /**/
    Currency c=new Euro();
    System.out.println(c);
    /**/
    return;
  }
}
```

Answer:

`$1.09`

This page intentionally is left blank.

6) Say that the class `Currency` from problem **5** is modified so that it implements the interface `Sortable` from problem **3** on page 4, *i.e.*, the lines `11 – 21` have been added, as shown below.

```
 1: public abstract class Currency implements Sortable {
 2:   /**/
 3:   public Currency () { }
 4:   /**/
 5:   public abstract double getUSDvalue ();
 6:   /**/
 7:   public String toString() {
 8:     return "$" + String.format("%4.2f",getUSDvalue());
 9:   }
10:   /**/
11:   public boolean lessThan ( Sortable s ) {
12:     /**/
13:     boolean rv;
14:     Currency otherCurrency;
15:     /**/
16:     otherCurrency=(Currency)s;
17:     if ( getUSDvalue() < otherCurrency.getUSDvalue() ) rv=true;
18:     else                                               rv=false;
19:     /**/
20:     return rv;
21:   }
22: }
```

The classes `CanadianDollar`, `Euro`, `Peso`, `PoundsSterling` and `USDollar`, as listed below, all extend `Currency`.

```
public class CanadianDollar extends Currency {
  public CanadianDollar ( ) { super(); }
  public double getUSDvalue() { return 0.75; }
}

public class Euro extends Currency {
  public Euro ( ) { super(); }
  public double getUSDvalue() { return 1.09; }
}

public class Peso extends Currency {
  public Peso ( ) { super(); }
  public double getUSDvalue() { return 0.05; }
}

public class PoundsSterling extends Currency {
  public PoundsSterling ( ) { super(); }
  public double getUSDvalue() { return 1.26; }
}

public class USDollar extends Currency {
  public USDollar ( ) { super(); }
  public double getUSDvalue() { return 1.00; }
}
```

We want to sort the currencies in the file `unsorted_currency.txt`, as listed here,

```
Canadian Dollar
Euro
Peso
Pounds Sterling
US Dollar
```

into ascending order, based on their monetary value, with class `Sort` from problem **3** on page 4. The program `SortCurrency.java`, listed on the next page, does this, *cf.*, line `8`.

```
 1: import java.util.ArrayList;
 2: /**/
 3: public class SortCurrency {
 4:   /**/
 5:   public static void main ( String [] arg ) {
 6:     /**/
 7:     ArrayList<Currency> currencyList=readListFromFile("unsorted_currency.txt");
 8:     Sort.<Currency>sort(currencyList);
 9:     printListToFile("sorted_currency.txt",currencyList);
10:     /**/
11:     return;
12:   }
13:   /**/
14:   public static ArrayList<Currency> readListFromFile ( String fn ) {
15:     /**/
16:     ArrayList<Currency> rv;
17:     FileInput fi;
18:     String line;
19:     /**/
20:     rv=new ArrayList<Currency> ();
21:     fi=new FileInput(fi);
22:     while ( !fi.eof() ) {
23:       line=fi.readLine();
24:       if      ( line.equals("Canadian Dollar") ) rv.add(new CanadianDollar());
25:       else if ( line.equals("Euro")            ) rv.add(new Euro()          );
26:       else if ( line.equals("Peso")            ) rv.add(new Peso()          );
27:       else if ( line.equals("Pounds Sterling") ) rv.add(new PoundsSterling());
28:       else if ( line.equals("US Dollar")       ) rv.add(new USDollar()      );
29:     }
30:     fi.close();
31:     /**/
32:     return rv;
33:   }
34:   /**/
35:   public static void printListToFile ( String fn, ArrayList<Currency> alc ) {
36:     /**/
37:     FileOutput fo;
38:     /**/
39:     fo=new FileOutput(fn);
40:     for ( Currency c : alc ) {
41:       fo.println(c);
42:     }
43:     fo.close();
44:     /**/
45:     return;
46:   }
47: }
```

When `SortCurrency.java` is run, what are the contents of the output file `sorted_currency.txt`?

Answer:

```
$0.05
$0.75
$1.00
$1.09
$1.26
```

7) If we wanted to sort the list in problem **6** into descending order, what change would need to be made to class `Currency`?

Answer:

Line `17` would need to be changed to
```
if ( getUSDvalue() > otherCurrency.getUSDvalue() ) rv=true;
```

This page intentionally is left blank.

8) Consider the class `IntList.java` which implements both of the interfaces `Iterable<Integer>` and `Iterator<Integer>`.

```
import java.util.Iterator;
/**/
public class IntList implements Iterable<Integer>, Iterator<Integer> {
  /**/
  private int [] intList;
  private boolean iteratorInitialized;
  private int position;
  /**/
  public IntList ( int [] a ) {
    intList=a;
    iteratorInitialized=false;
    position=-1;
  }
  /**/
  public Iterator<Integer> iterator () {
    iteratorInitialized=true;
    position=0;
    return this;
  }
  /**/
  public boolean hasNext() {
    if ( !iteratorInitialized ) error();
    if ( position != intList.length ) return true;
    else                              return false;
  }
  /**/
  public Integer next () {
    if ( !iteratorInitialized ) error();
    return new Integer(intList[position++]);
  }
  /**/
  private static void error () {
    throw new IllegalStateException ("Iterator has not been initialized.");
  }
}
```

The program `IntListTest.java`, listed here, uses class `IntList` (*e.g.*, on line `8`).

```
 1: import java.util.Iterator;
 2: /**/
 3: public class IntListTest {
 4:   /**/
 5:   public static void main ( String [] arg ) {
 6:     /**/
 7:     int [] ia=new int [] { 1 , 2 , 3 };
 8:     IntList il=new IntList(ia);
 9:     /**/
10:     int option=Integer.parseInt(arg[0]);
11:     /**/
12:     if      ( option == 1 ) {
13:       Iterator<Integer> i=il.iterator();
14:       while ( i.hasNext() ) System.out.println( i.next() );
15:     }
16:     else if ( option == 2 ) {
17:       for ( Integer i : il ) System.out.println( i );
18:     }
19:     else if ( option == 3 ) {
20:       for ( int i : ia ) System.out.println( i );
21:     }
22:     return;
23:   }
24: }
```

For which invocation of the program, *viz.*, `java IntListTest 1`, `java IntListTest 2` or `java IntListTest 3`, will the output of the program be as follows?

```
1
2
3
```

Answer:

All three invocations.

9) What is the approximate order of the number of operations which will be performed when a list of *n* objects is sorted with the insertion sort algorithm?

Answer:

$O\ (n^2)$

10) What is the resulting output when the program `DoubleIt.java` is run?

```
public class DoubleIt {
  /**/
  private int k;
  /**/
  public static void main ( String [] arg ) {
    DoubleIt di=new DoubleIt();
    di.doubleK(7);
    System.out.println(di.getNumber());
    return;
  }
  /**/
  public DoubleIt ( ) {
    k=1;
  }
  /**/
  public int getNumber() { return k; }
  /**/
  public void doubleK ( int aNumber ) {
    if ( aNumber > 1000 )  return;
    k=aNumber;
    k *= 2;
    doubleK(k);
  }
}
```

Answer:

1792

Problems **11** through **13** concern the program `MessWithAnArray.java`.

```
public class MessWithAnArray {
  /**/
  public static void main ( String [] arg ) {
    /**/
    int [] a=new int [] { 1 , 2 };
    int [] b=null;
    /**/
    int option=Integer.parseInt(arg[0]);
    if      ( option == 1 ) {
      b=new int [2];
      b[0]=a[1];   b[1]=a[0];
    }
    else if ( option == 2 ) {
      b=a;
      b[0]=a[1];
    }
    else if ( option == 3 ) {
      b=a;
      b[1]=a[0];
    }
    /**/
    String outS = Integer.toString(a[0])
                + Integer.toString(b[0])
                + Integer.toString(a[1])
                + Integer.toString(b[1]);
    System.out.println(outS);
    /**/
    return;
  }
}
```

11) What is the resulting output if the program is invoked via `java MessWithAnArray 1`?

Answer:

`1221`

Here the arrays `a` and `b` are separate objects.

12) What is the resulting output if the program is invoked via `java MessWithAnArray 2`?

Answer:

`2222`

Here the arrays `a` and `b` are the same object.

13) What is the resulting output if the program is invoked via `java MessWithAnArray 3`?

Answer:

`1111`

Here the arrays `a` and `b` are the same object.

This page intentionally is left blank.

14) Consider the file `unsorted_doubles.txt`, the first few lines of which are listed below, and which file contains random floating-point numbers *n* such that $0.00 \leq n \leq 99.99$.

```
15.40
92.44
90.08
27.05
49.61
 9.04
  .
  .
  .
```

We want to sort this list with the class `PerformSort.java`, listed here,

```
 1: import java.util.ArrayList;
 2: /**/
 3: public class PerformSort <T extends Sortable<T>> {
 4:   /**/
 5:   private ArrayList<T> alt;
 6:   private ArrayList<T> work;
 7:   /**/
 8:   public PerformSort ( ArrayList<T> alt ) {
 9:     this.alt=alt;
10:     work=new ArrayList<T> ();
11:     for ( int i=0; i<alt.size(); ++i ) work.add(alt.get(i));
12:   }
13:   /**/
14:   public void performSort ( ) {
15:     /**/
16:     int len, lo, hi;
17:     /**/
18:     len=alt.size();
19:     lo=0;   hi=len-1;
20:     partition(lo,hi);
21:     /**/
22:     return;
23:   }
24:   /**/
25:   private void partition ( int lo, int hi ) {
26:     /**/
27:     int mid, top, bot;
28:     T pivot, ai;
29:     /**/
30:     if ( lo >= hi ) return;
31:     /**/
32:     mid=(lo+hi)/2;
33:     pivot=alt.get(mid);
34:     top=lo;   bot=hi;
35:     for ( int i=lo; i<=hi; ++i ) {
36:       ai=alt.get(i);
37:       if      ( ai.lessThan(pivot) ) work.set(top++,ai);
38:       else if ( pivot.lessThan(ai) ) work.set(bot--,ai);
39:     }
40:     work.set(top,pivot);
41:     for ( int i=lo; i<=hi; ++i ) alt.set(i,work.get(i));
42:     /**/
43:     partition(lo,top-1);
44:     partition(top+1,hi);
45:     /**/
46:     return;
47:   }
48: }
```

which class sorts a list `ArrayList<T>` of `T` objects into ascending order, where class `T` implements the `Sortable<T>` interface, listed here.

```
public interface Sortable<T> {
  boolean lessThan ( T t );
}
```

In order to sort our list of floating-point numbers, we need a class which implements `Sortable<T>`. The class `MyDouble.java`, listed below, does just that.

```
public class MyDouble implements Sortable<MyDouble> {
  /**/
  private double d;
  /**/
  public MyDouble ( double aDouble ) { d=aDouble; }
  /**/
  public double getD () { return d; }
  public String toString () { return String.format("%5.2f",d); }
  /*
   *  As per interface Sortable<MyDouble>
   */
  public boolean lessThan ( MyDouble otherD ) {
    if ( d < otherD.getD() ) return true;
    else                     return false;
  }
}
```

Finally, here is the class which ties all of this together, `SortDoubles.java`.

```
import java.util.ArrayList;
/**/
public class SortDoubles {
  /**/
  public static void main ( String [] arg ) {
    /**/
    ArrayList<MyDouble> list=readFromFile("unsorted_doubles.txt");
    PerformSort ps=new PerformSort(list);
    ps.performSort();
    printToFile(list,"sorted_doubles.txt");
    /**/
    return;
  }
  /**/
  public static ArrayList<MyDouble> readFromFile ( String fileName ) {
    /*
     *  Implementation not shown
     */
  }
  /**/
  public static void printToFile ( ArrayList<MyDouble> almd, String fileName ) {
    /*
     *  Implementation not shown
     */
  }
}
```

What is the sorting algorithm implemented by class `PerformSort.java` called?

Answer:

Quick sort

15) What is the approximate order of the number of operations required to sort a list of *n* objects with the quick sort algorithm?

Answer:

$O\ (n \log n)$

Problems **16** and **17** concern the program `IntTest.java`. Recall the an `int` requires 4 bytes of storage.

```
 1: public class IntTest {
 2:   /**/
 3:   public static void main ( String [] arg ) {
 4:     /**/
 5:     int a, b, c, d;
 6:     /**/
 7:     a=0x7fffffff;
 8:     b=a+1;
 9:     c=0x80000000;
10:     d=c-1;
11:     /**/
12:     System.out.println();
13:     System.out.println( "  a = " + a );
14:     System.out.println( "a+1 = " + b );
15:     System.out.println( "  c = " + c );
16:     System.out.println( "c-1 = " + d );
17:     /**/
18:     return;
19:   }
20: }
```

16) What is the resulting output from lines `13` and `14` of the program?

Answer:

```
  a = 2147483647
a+1 = -2147483648
```

Note that this is due to overflow.

17) What is the resulting output from lines `15` and `16` of the program?

Answer:

```
  c = -2147483648
c-1 = 2147483647
```

Again, this is due to overflow.

18) For the built-int (primitive) integer type `long`, the maximum value is `Long.MAX_VALUE` $= 2^{63} - 1$, and the minimum value is `Long.MIN_VALUE` $= -2^{63}$. How many bytes of storage does a `long` occupy?

Answer:

The number of integers is $2^{64} \Longrightarrow 64$ bits $\div 8 \Longrightarrow 8$ bytes.

19) An array `la` is instantiated via `long [] la=new long [100000]`. Excluding the storage required for the `long []` pointer, how many kbytes does `la` occupy?

Answer:

800,000 bytes $\div 1024 \Longrightarrow 781.25$ kbytes.

20) Consider the program `ListOfDoubles.java`.

```
public class ListOfDoubles {
  /**/
  private double [] list;
  /**/
  public static void main ( String [] arg ) {
    ListOfDoubles lod=new ListOfDoubles(100);
    lod.printListToFile("list.txt");
    return;
  }
  /**/
  public ListOfDoubles ( int num ) {
    list=new double [num];
    init();
  }
  /**/
  public void printListToFile ( String fileName ) {
    FileOutput fo=new FileOutput(fileName);
    for ( double d : list ) fo.println( String.format("%5.2f",d) );
    fo.close();
    return;
  }
  /**/
  private void init () {
    /**/
    int n;
    double rd;
    /**/
    n=list.length;
    for ( int i=0; i<n; ++i ) list[i]=(double)n;
    for ( int i=0; i<n; ++i ) {
      while ( true ) {
        rd=(double)n*Math.random();
        if ( !contains(rd) ) break;
      }
      list[i]=rd;
    }
    /**/
    return;
  }
  /**/
  private boolean contains ( double n ) {
    int len=list.length;
    for ( int i=0; i<len; ++i ) {
      if ( list[i] == n ) return true;
    }
    return false;
  }
}
```

What does the output file `list.txt` contain upon running the program?

Answer:

A list of floating-point numbers between `0.00` and `99.99` (inclusive) ordered randomly.