QUESTION 1.

SHORT QUESTIONS (5 Points each part = 50 Points).

(a) (5 Points for each error)
This Java program has two errors.

```
1:   public class FixMe {
2:     int nn;
3:     nn = 111;
4:     public static void main (String[] args) {
5:         System.out.printf("nn = %d\n", nn);
6:     }
7: }
```

Describe the two errors. For each error, state whether it is a run-time or compile-time error.

**SOLUTION:** First Error: Lines 2 and line 5 together results in an error because you cannot access the instance variable `nn` from a static method like `main`. It is a compile-time error.
Most students caught this one. Note that this error is distributed in lines 2 and 5, but we should not count this as two errors – each is fine by itself.
Second Error: the assignment in line 3 is not valid because such assignments must either be part of an initialization or inside some method. This is a compile-time error.
Most students do not understand this error.

**Comments:** Many students often say these are run-time errors. All these are "syntax errors" and therefore these programs cannot even be compiled (so there is no chance of being run to result in a run-time error). Also, students are not familiar with the printf format and think there are errors here. Note that we use this format extensively in our sample programs.

(b) Please remove both errors in `FixMe`

**SOLUTION:** Two ways to fix the program (a) You can declare `nn` to be static, and also initialize `nn` to 111 directly:

```
int nn=111;
```

(b) Alternatively, we can access `nn` via an instance of `FixMe`:

```
public static void main (String[] args) {
    FixMe fm = new FixMe();
    fm.nn = 111;
    System.out.printf("nn = %d\n", fm.nn);
}
```
(c) Using the geometric shape examples from Chapter 1, consider:

```java
public void changeParameter (Circle c) {
    c.setRadius(5);
}

public static void main(String[] args) {
    Circle a = new Circle();
    a.setRadius(10);
    Circle b = a;
    changeParameter(b);
}
```

What are the radii of circle `a` and circle `b` at the end?

**SOLUTION:** Both have radius 5.

(d) Suppose we have doubly-linked lists whose nodes are instances of

```java
class Node {
    String val;
    Node next, prev;
}
```

Write the method `void remove(Node u)` which will remove the node that precedes `u`. If `u` or `u.prev` is null, then this code does nothing.

**SOLUTION:**

```java
void remove(Node u) {
    if (u==null || u.prev==null) return;
    u.prev = u.prev.prev;
    if (u.prev != null) u.prev.next = u;
}
```

(e) Suppose the running time for an algorithm satisfies this recurrence

\[
T(n) = \begin{cases} 
4 & \text{for } n \leq 2 \\
10 + 3T(n-1) & \text{else.}
\end{cases}
\]

Prove that \(T(n) = 3^n - 5\) is a solution for \(n \geq 2\).

**SOLUTION:** Simple induction: BASE CASE: We verify that \(T(2) = 3^2 - 5 = 4\).

INDUCTION: For \(n \geq 3\), \(T(n) = 10 + 3(3^{n-1} - 5) = 10 + 3^n - 15 = 3^n - 5\).

(f) Here is a partial listing of my Makefile:
jcc:
   ---> test -d bin || mkdir bin
   ---> -javac -d bin src/**/*.java

hw1:
   ---> make p=hw1 hw1-helper

hw1-helper:
   ---> java src/hw1/myProg $(args)

As usual ---> just indicates a tab. Describe all the actions of the jcc target.

**SOLUTION:** It will create a bin directory if none exists. Then the javac command will compile all the java programs in subfolders of src. The class files are put into the bin folder. The symbol "-" in front of javac tell Makefile not to consider it an error if there are no java files in a subfolder of bin! Otherwise, the make program will stop at the first error.

**Comments:**

(g) In the Makefile above, what is the point of target hw1 calling another target hw1-helper? Why not call hw1-helper directly? HINT: our two goals are to set up default arguments that depend upon the target (hw1, hw2, etc) and the ability to override these values from the terminal.

**SOLUTION:** This construction allows us to set up default values for args, but it also allows us to override these values at the terminal. When we call hw1, it will set the variable p=hw1 for a recursive call to make. Elsewhere in the Makefile (not shown), we can test the value of p in order to set up the default values for args depending on where p=hw1 or something else. E.g.,

```make
ifeq ($(p),hw1)
   nn=20
   mm=1
   args= $(nn) $(mm)
endif
```

You might think: why don’t we just set up the default value of args directly in hw1? E.g., if the default value of args is 20, you might just do:

```
hw1:
   ---> java src/hw1/myProg 20
```

The problem is that you cannot, in the command line, decide to replace 20 by 30. But using our above design, we can type at the terminal

```
> make hw1 nn=30
```

will allow us to override the default value of nn (but keep the default value of mm).

**Comments:** HOW does this all work? When make program is initially called, it has a set of variables that are already assigned values (it inherits this from the shell environment). The set of all these variable=value pairs is known as the "environment". After the first call to make, make will add additional variable=value pairs to the environment. I.e., its environment is expanded. So when it calls itself again, it begins with this expanded environment. Any value we set at the command terminal when we first call "make" will remain in the expanded environment!
(h) We want a pattern that is matched by names that are written following certain common conventions (assuming only two-part names). More precisely: write any pattern that is matched by these three strings

“Firstname Lastname”, “First Last”, “Chee Yap”

but not by any of these five strings

“First Middle Last”, “First LAST”, “First Last “, “CheeYap”, “first last”

Note that we allow spaces before the name, but not after the name.

SOLUTION:

"\s*[A-Z][a-z]*\s+[A-Z][a-z]*\$"

Comments: Note that we use the $ symbol to match the end of string, similar to the caret character to match the beginning of string.

(i) Write a Java pattern expression to convert the string

ss = "hw1.java hw2.javac zoombini.java fibjava java.class fib.java"

into

"hw1.class hw2.javac zoombini.class fibjava java.class fib.class"

SOLUTION: The desired expression is

ss.replaceAll(pat,rep)

where

String pat="([a-z0-9]+)\.(java)(\s|$)"
String rep="$1.class$3"

Comments: Note that we have 3 groups in our pattern. The third group is necessary to avoid turning "hw2.javac" into hw2.class”. It says that we must terminate the "java" with a space or end of string, and this group is reproduced in the replacement pattern.

QUESTION 2. (2+10+4+12+20 = 47 Points)

The function $G(n)$ is recursively defined by the formula

$$G(n) = \begin{cases} 
1 & \text{for } n \leq 2 \\
G(n-1) + G(n-2) + G(n-3) & \text{else.}
\end{cases}$$

Points are deducted for Java syntax errors in the programming parts.

(a) What is $G(6)$?

SOLUTION: $G(6) = 17$.

(b) Implement a non-exponential time method to compute $G(n)$ based on recursion. Please use the following header for your method:

```java
BigInteger G(int n);
```

HINT: call a helper method to do the recursive work.
SOLUTION: Here is the recursive solution:

```java
static BigInteger base =
    {BigInteger.ONE, BigInteger.ONE, BigInteger.ONE};

BigInteger G(int n) {
    return G(n, base);
}

BigInteger G(int n, base) {
    if (n<=2) return base[0];
    BigInteger tmp = base[0].add(base[1]);
    base[2] = base[1];
    base[1] = base[0];
    base[0] = tmp;
    return G(n-1, base);
}
```

Comments: Note that this is a tail recursion, so we could have written it as a while- or for-loop.

(c) What is the big-Oh running time of your method?

SOLUTION: The running timing of $G(n)$ is $O(n)$.

(d) The value $G(n)$ will be a very large integer even for modest values of $n$. We would like to only see the last few digits of the number (this is enough to visually check that our computation is correct). To accomplish this, write the following method

```java
String truncate(BigInteger N)
```

that returns a String of length at most 5, representing the last 5 digits of the integer $N$. In case $N$ has more than 5 digits, your output string must be preceded by 3 asterisks, as in `***12345`. HINT: $N$.remainder($M$) will return the remainder of $N$ divided by the BigInteger $M$.

SOLUTION:

```java
String truncate(BigInteger N) {
    BigInteger M = new BigInteger("100000");
    if (N.compareTo(M)<=0) return N.toString();
    return ("***" + N.remainder(M).toString();
}
```

(e) Write a class called GG containing the above methods. Include a main method that takes two optional command line arguments: `int nn, int mm`. Their default values are `nn=1000, mm=4`. The goal is to compute a table of $mm$ consecutive values of the function $G(n)$, starting from $n$ equal to $nn$. Try to format the output table nicely. This problem requires file manipulation because you are to write your table into a file called GTABLE.
QUESTION 3. (3+5+8+24=40 Points) Here is the stack interface from the textbook:

```java
public interface StackInterface<T> {
    void pop() throws StackUnderflowException;
    T top() throws StackUnderflowException;
    boolean isEmpty();
}
```

(a) Explain why the push operation is missing in this interface for a stack?

**SOLUTION:** The reason is that the push operation might behave differently depending on whether we use an array or a linked list to implement the stack. So we do not want to commit ourselves to which version of stacks we want.

(b) Write an extension of `StackInterface` with a push method that is suitable for an linked list implementation.

**SOLUTION:** We may call this interface `UnboundedStackInterface` and it extends the `StackInterface` with just one additional method,

```java
void push(T item);
```

(c) Write an extension of `StackInterface` with a push method that is suitable for an array implementation. Besides adding a push method, think of an additional method that would be useful to have in your extension.

**SOLUTION:** We may call this interface `BoundedStackInterface` and it extends the `StackInterface` with just one additional method,

```java
void push(T) throws StackOverflowException;
```
Since the array can be full, it would be nice to add a method to check this condition:

```java
boolean isFull();
```

Comments:

(d) Write a class called `MyLinkedStack` that implements your interface in part (b) with `T=String`. You must include a main method (no need for any command line arguments). The method is to do some pushes and pops of random strings, in order to exercise all your methods. How to get the random strings? Generate random integers and convert them into strings. Again, do not forget any needed imports.

PLEASE USE THE BACK OF THIS PAGE TO WRITE YOUR CODE.
SOLUTION: We assume our usual Node class with implements a linked list. But we write it here as an inner class:

```java
import java.util.Random;

class MyLinkedStack implements UnboundedStackInterface<String> {
    Node bot = new Node("");
    Node tos = bot; // tos=Top-of-Stack
    class Node {
        String val; Node next;
    }

    public void pop() throws StackUnderflowException {
        // leave unimplemented -- use top instead
    }

    public String top() throws StackUnderflowException {
        if (tos == bot) throw new StackUnderflowException("top");
        return tos.val;
    }

    public boolean isEmpty() { return (tos==bot); }

    public void push(String e) {
        Node t = new Node(e);
        t.next = tos;
        tos = t;
    }

    public static void main(String[] args) {
        Random rg = new Random();
        StackString ss = new StackString();

        for (int i=0; i<20; i++)
            ss.push("" + rg.nextInt(100));
        for (int i=0; i<20; i++) {
            System.out.printf("%s ", ss.top());
            ss.pop();
        }
        if (ss.isEmpty()) System.out.println("Empty!");
    }
}
```