1 Definitions (15 points)

Define each of the following terms.

a. abstraction barrier - separates the caller from the internal implementation of an abstraction
b. garbage - objects that are no longer accessible
c. accessor - a method that provides access to information held within an object
d. base case - in a recursive method, the case in which no further recursive calls are required because the result can be directly determined

e. class - a type of object

f. (for extra credit from Undergraduate Research Day) dispatching - the act of initiating execution of a method or process

2 Procedure Comprehension (30 points)

a. (8 points) The following method is incorrect. It is supposed to use successive subtraction to determine if the parameter \( m \) is divisible by the parameter \( n \) (with no remainder). Show how to modify the method so that it works properly. Assume that the parameter values are positive integers.

```java
boolean isDivisible(int m, int n) {
    if (m < n)
        return false;
    else if (m == n)
        return true;
    return isDivisible(m - n, n);
}
```

b. (6 points) At the right, show the evaluation of \( \text{foo}(3) \) using the substitution model.

```java
int foo(int i) {
    if (i == 0)
        return 1;
    else
        return 2 * foo(i-1);
}
```

\[
\begin{align*}
\text{foo}(3) & = 2 \times \text{foo}(2) \\
2 & \times \text{foo}(1) \\
2 & \times 2 \times \text{foo}(0) \\
2 & \times 2 \times 2 \times 1 = 8
\end{align*}
\]

Assuming the parameter value is always non-negative, what would be a better name for \( \text{foo} \)?

(That is, what does it compute?) twoToTheN

3 pts.
c. (8 points) Recall that the Line constructor takes as parameters the coordinates of the endpoints \((x_1,y_1)\) and \((x_2,y_2)\) of the line segment to be drawn. Draw the picture that would be produced by the following method. **On your drawing, label the coordinates of the endpoints of each line.**

```java
void draw(GraphicsPanel gp) {
    int a = 50;
    int b = 100;
    while (a < b) {
        gp.add(new Line(a,b,b,a));
        gp.add(new Line(a,a,b,b));
        a = a + 30;
    }
}
```

50, 100, 100, 50
50, 50, 100, 100
80, 100, 100, 80
80, 80, 100, 100

---

d. (8 points) Suppose we let int[] a = \{1, 2, 3, 4, 5\}. What is in the array a after reverse(a) executes? (Hint: This is a trick question.)

```java
void reverse(int[] a) {
    for (int k = 0; k < a.length; k++) {
        a[k] = a[a.length - 1 - k];
    }
}
```

\(a = \{5, 4, 3, 4, 5\}\)

\((-4, f(5, 4, 3, 2, 1))\)
3 Object Comprehension (30 points)

Read following Java code carefully and then answer the questions. The code compiles, but it might not behave as you would expect.

```java
public class Account {
    int balance;

    public Account(int initialBalance) {
        int balance = initialBalance;
    }
    public int deposit(int amount) {
        if (amount < 0)
            throw new IllegalArgumentException("negative amount");
        balance = balance + amount;
        return balance;
    }
    public int withdraw(int amount) {
        if (amount > balance)
            throw new IllegalArgumentException("insufficient funds");
        balance = balance - amount;
        return balance;
    }
    public void transfer(int amount, Account destination) {
        withdraw(amount);
        destination.deposit(amount);
    }
    public Account merge(Account other) {
        Account result = new Account(0);
        transfer(balance, result);
        other.transfer(other.balance, result);
        return result;
    }
    public String toString() { return balance + ".00"; }
}
```

a. Does the following test succeed or fail? **fails** If it fails, explain why.

Account a = new Account(50); ← the constructor doesn't initialize the balance
assertEquals("50.00", a.toString()); ← fails here because expecting "0.00"

b. Does the following test succeed or fail? **succeeds** If it fails, mark the line where it fails and explain why it fails.

Account a = new Account(0);
assertEquals(50, a.deposit(50));

c. Does the following test succeed or fail? **fails** If it fails, mark the line where it fails and explain why it fails.

Account a = new Account(0);
assertEquals(-50, a.deposit(-50)); ← IllegalArgument Exception thrown by deposit (negative amount)
d. Does the following test succeed or fail? **succeeds** If it fails, mark the line where it fails and explain why it fails.

```java
Account a = new Account(0);
try {
    assertEquals(-50, a.deposit(-50));
    fail();
} catch (IllegalArgumentException iae) {
}
```

e. Does the following test succeed or fail? **fails** If it fails, mark the line where it fails and explain why it fails.

```java
Account a = new Account(0);
assertEquals(-50, a.withdraw(50)); ← Illegal Argument Exception (insufficient funds)
```

f. Does the following test succeed or fail? **succeeds** If it fails, mark the line where it fails and explain why it fails.

```java
Account a = new Account(0);
assertEquals(50, a.withdraw(-50));
```

g. Does the following test succeed or fail? **fails** If it fails, mark the line where it fails and explain why it fails.

```java
Account a = new Account(0);
Account b = new Account(0);
a.deposit(50);
a.transfer(-50, b); ← Illegal Argument Exception when trying to deposit -50 into b
assertEquals("100.00", a);
assertEquals("-50.00", b);
```

h. Does the following test succeed or fail? **fails** If it fails, mark the line where it fails and explain why it fails.

```java
Account a = new Account(0);
Account b = a;
a.deposit(50);
a.transfer(50, b);
assertEquals("0.00", a);
assertEquals("50.00", b);
```

← expecting "50.00" (a + b refer to the same object)
i. Does the following test succeed or fail? **succeeds** If it fails, mark the line where it fails and explain why it fails.

```java
Account a = new Account(0);
Account b = new Account(0);
a.deposit(50);
b.deposit(100);
Account c = a.merge(b);
assertEquals("0.00", a.toString());
assertEquals("0.00", b.toString());
assertEquals("150.00", c.toString());
```

j. Does the following test succeed or fail? **succeeds** If it fails, mark the line where it fails and explain why it fails.

```java
Account a = new Account(0);
Account b = new Account(0);
a.deposit(50);
b.withdraw(-50);
Account c = a.merge(b);
assertEquals("0.00", a.toString());
assertEquals("0.00", b.toString());
assertEquals("100.00", c.toString());
```

4 Recursion and Iteration (25 points)

For integer $x \geq 0$, let

$$f(x) = \begin{cases} 
0 & \text{if } x = 0 \\
x & \text{if } x \text{ is odd} \\
f(x-1) + f(x-2) & \text{otherwise}
\end{cases}$$

So the series is:

$$x = \begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & \\
f(x) = 0 & 1 & 1 & 3 & 4 & 5 & 9 & 7 & 16 & 9 & 25 & 
\end{array}$$

a. (10 points) Use recursion to write a correct Java method that takes as input an integer $x > 0$ and returns the value of $f(x)$. Do not use a loop.

```java
int foo(int x) {
    if (x == 0)
        return 0;
    else if (x % 2 == 1)
        return x;
    else
        return foo(x-1) + foo(x-2);  \text{could just be the expression } x-1 \text{ to save time}
}
```
b. (10 points) Write a correct **iterative** Java method that it takes as input an integer $x > 0$ and returns $f(x)$. Do **not** use recursion. You may create as many loop variables as you like, but do not change the value of the parameter $x$.

```java
int foo(int x) {
    int f_k = 1;
    int f_k_1 = 0;
    int k = 1;
    while (k < x) {
        k++;
        int previous = f_k_1;
        f_k_1 = f_k;
        if (k % 2 == 1)
            f_k = k;
        else
            f_k = f_k + previous;
    }
    return f_k;
}
```

Solutions will vary.

c. (5 points)

1. State a **loop invariant** that holds for the procedure you completed in part b.

```latex
\begin{align*}
\text{f}_k &= f(k) \quad \& \quad \text{f}_{k-1} = f(k-1) \\
\text{some students may not cover this part (all loop vars)}
\end{align*}
```

2. Argue that your loop invariant, together with the termination condition, implies the correctness of the return value.

```latex
\begin{align*}
\text{On termination, } k &= x \\
\text{Substituting into the invariant, } \text{f}_{k} &= f(k) = f(x) \checkmark
\end{align*}
```

d. (1 point extra credit) Write a correct Java method that it takes as input an integer $x > 0$ and returns $f(x)$. Use **neither** recursion nor iteration. You may use a conditional (if) statement.

```java
int foo(int x) {
    if (x % 2 == 1)
        return x;
    else
        return (x / 2) * (x / 2);
}
```