Directions:

- This exam is closed book and closed notes. Do not use a calculator or any other computing device.

- Please check that you have all pages, numbered 1 through 7. Write your name on each piece of paper, in case the pages become separated.

- Write your answers concisely and legibly directly on this exam. Do not use your own paper. In case you need to, you can use the back of page 7 as scratch paper.

- Do your own work. No discussion or collaboration with other students is permitted.

- If a question is unclear to you, please walk to the front of the room and ask me about it quietly.

- The exam is divided into four parts: definitions (15 points), procedure comprehension (30 points), object comprehension (30 points), and recursion and iteration (25 points).

- If you get stuck on a question, don’t waste too much time on it. Go on to the next question and the answer may occur to you later.

- Partial credit will be given where appropriate. If you see how to approach a problem but don’t see the final answer, be sure to at least write down your approach.

<table>
<thead>
<tr>
<th>Section</th>
<th>Score</th>
<th>Possible</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Definitions</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>2. Procedure Comprehension</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>3. Object Comprehension</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>4. Recursion and Iteration</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
1 Definitions (15 points)

Define each of the following terms.

a. *abstraction barrier* –

b. *garbage* –

c. *accessor* –

d. *base case* –

e. *class* –

f. (for extra credit from Undergraduate Research Day) *dispatching* –

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-m);
}
```

b. (6 points) At the right, show the evaluation of `foo(3)` using the substitution model.

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

Assuming the parameter value is always non-negative, what would be a better name for `foo`? (That is, what does it compute?)

[Write in answer: ________________________________]
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;
    }
}
```

d. (8 points) Suppose we let \(\text{int}[\] a = \{1, 2, 3, 4, 5\}. What is in the array \(a\) after \(\text{reverse}(a)\) 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];
    }
}
```
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) {
        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? ____________ If it fails, explain why.
   ```java
   Account a = new Account(50);
   assertEquals("50.00", a.toString());
   ```

b. Does the following test succeed or fail? ____________ If it fails, mark the line where it fails and explain why it fails.
   ```java
   Account a = new Account(0);
   assertEquals(50, a.deposit(50));
   ```

c. Does the following test succeed or fail? ____________ If it fails, mark the line where it fails and explain why it fails.
   ```java
   Account a = new Account(0);
   assertEquals(-50, a.deposit(-50));
   ```
d. Does the following test succeed or fail? ____________ 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? ____________ 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));
```

f. Does the following test succeed or fail? ____________ 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? ____________ 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);
assertEquals("100.00", a);
assertEquals("-50.00", b);
```

h. Does the following test succeed or fail? ____________ 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);
```
i. Does the following test succeed or fail? 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? 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:

\[
\begin{align*}
x &= 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ \ldots \\
f(x) &= 0 \ 1 \ 1 \ 3 \ 4 \ 5 \ 9 \ 7 \ 16 \ 9 \ 25 \ \ldots
\end{align*}
\]

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.
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 $x$.

c. (5 points)

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

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

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.