Due at end of session

This exam is: closed-book, NO electronic devices allowed, and closed-notes. The exception is the “sage page” of the designated size on which you may have notes to consult during the exam.

Be sure you: Provide legible answers in designated areas (credit will not be given for work that is difficult to read or not where expected), Ensure you clearly fill in a single circle on multiple choice questions, Use indentation of your code to show its structure (but don’t dwell on exact punctuation/syntax), Leave the exam stapled together in its original order, Do NOT attach any other pages to the exam. You are welcome to use the blank space on the exam for any scratch work.

If you need to leave the room for any reason prior to turning in your exam, you must leave your exam and any electronic devices with a proctor.

You must complete all the identifying information below correctly. Failure to do so is grounds for a zero on this exam:

1. Name (print clearly):

2. Student ID (print clearly; 1 digit per underline): ___ ___ ___ ___ ___ ___ ___

3. Which time do you typically attend studio (fill one)
   - [ ] 11:30—1:00
   - [ ] 1:00—2:30
   - [ ] 2:30—4:00
   - [ ] 4:00—5:30

4. Which Urbauer lab do you typically sit in (best guess; fill one)
   - [ ] 214
   - [ ] 216
   - [ ] 218
   - [ ] 222

5. You must sign the pledge below for your exam to count. The penalty for cheating will be decided during academic integrity review, but the instructors will recommend an F in this course as the minimum penalty.

   I have read the instructions on this page and I will neither give nor receive any unauthorized aid on this exam.

(Sign above)

⇒ Do not proceed until told to do so! ⇐

⇒ Initial the top right corner of each page before starting ⇐
1. (4 points) Indicate whether each is primarily considered an Input or an Output device:

(1) Keyboard
   - Input [ ]
   - Output [ ]

(2) Video Display Projector
   - Input [ ]
   - Output [ ]

(3) Mouse
   - Input [ ]
   - Output [ ]

(4) Printer
   - Input [ ]
   - Output [ ]

2. (6 points) True or False:

(1) A method’s signature includes:
   i. The method’s name [ ]
   ii. The method’s parameters’ names [ ]
   iii. The method’s parameters’ types [ ]
   iv. The method’s body [ ]

(2) Methods can have more than one return statement in their body:
   - True [ ]
   - False [ ]

(3) All methods must have at least one return statement:
   - True [ ]
   - False [ ]

3. (3 points) Fill in the blank with the most appropriate term:

(1) A method that calls itself is an example of ____________
   and the special situation(s) where it does not call itself is/are referred to as the ____________.

(2) Java uses a ____________ to store parameter values, return values, and local variables and to allow methods to call one another.

4. (1 point) Methods are a form of abstraction. Someone who only knows the details of the method’s signature can use the method successfully. The other details, such as the actual code and its local variables, are “hidden”. This is referred to as:
   - Cronification [ ]
   - Encapsulation [ ]
   - Exoneration [ ]
   - Privitization [ ]
   - Sequesteration [ ]
5. (8 points) In each of the following snippets of code circle the code corresponding to each term:

(1) The **Return Type**

```java
public static String slice(String in, int b, int e) {
    String s=
    for(int i=b; i<e; ++i) {
        s = s + in.charAt(i);
    }
    return s;
}
```

(2) The **Method Name**

```java
public static String slice(String in, int b, int e) {
    String s="
    for(int i=b; i<e; ++i) {
        s = s + in.charAt(i);
    }
    return s;
}
```

(3) The **Body**

```java
public static String slice(String in, int b, int e) {
    String s="
    for(int i=b; i<e; ++i) {
        s = s + in.charAt(i);
    }
    return s;
}
```

(4) The **scope of i**:

```java
public static String slice(String in, int b, int e) {
    String s="
    for(int i=b; i<e; ++i) {
        s = s + in.charAt(i);
    }
    return s;
}
```
6. (5 points) Each method described below will process an array of words. Fill in the most appropriate choice for the missing parts of the signature:

(1) `printWords` will print all the words to the console (via `System.out.println`).

```
public static ___________ printWords(___________ words)
```

(2) `avgLength` computes the average length using all the words (but does not print it itself).

```
public static ___________ avgLength(___________ words)
```

7. Given the following two methods, show the output of the piece of code shown on the following. Be as accurate as possible, including spacing and line usage. Output should start on the first dotted line on the right side of the paper (the leftmost dot corresponds to the left side of the console window).

```
public static int a(int x) {
    System.out.println("A");
    return x + 3;
}
public static int b(int y) {
    System.out.println("B");
    y = y + 1;
    return y;
}
```

(1) (6 points)
```
int y = 3;
int m = b(y);
System.out.println(m);
System.out.println(y);
```

...............................
...............................
...............................
...............................
...............................
...............................
8. This is a continuation of the previous problem. Methods a() and b() are the same as they were in the last problem, but c() and d() are new. You should follow the same format.

```java
public static int a(int x) {
    System.out.println("A");
    return x + 3;
}

public static int b(int y) {
    System.out.println("B");
y = y + 1;
    return y;
}

public static int c(int z) {
    System.out.println("C");
    int w = a(z);
    w = w + b(z);
    return w;
}

public static int d(int w) {
    System.out.println("D");
    System.out.println(c(w));
    return w * 5;
}
```

(1) (4 points)
```
int n = c(3);
System.out.println(n);
```

(2) (6 points)
```
System.out.println(d(1));
```
9. Given the following methods show the output of each specific call shown below. Be as accurate as possible, including spacing and line usage. Output should start on the first dotted line on the right side of the paper (the leftmost dot corresponds to the left side of the console window).

```java
public static void m(int x, int y) {
    System.out.println("x:" + x);
    if (y>3) {
        m(x,y-1);
        m(x-2,y);
    } else if (x>0) {
    }
}
```

(1) (2 points)
m(-2, -1); ............................... ...............................

(2) (4 points)
m(2, 0); ............................... .............................. ..............................

(3) (6 points)
m(2, 4); ............................... .............................. .............................. ..............................
The method below is a slight variation on the previous one. The location of the println has been changed. Show the output of each specific call shown below (note that the cases are identical to some of those on the previous problem).

```java
public static void m2(int x, int y) {
    if (y>3) {
        m2(x, y-1);
    } else if (x>0) {
        m2(x-2, y);
    }
    System.out.println("x:" + x);  // println moved here!
}
```

(4) (2 points)

m2(2, 0);

(5) (3 points)

m2(2, 4);
10. (6 points) The Nascent Forensics Laboratory (NFL) is investigating The Case of the Cold, Flat Tire. Tragically, no one at the NFL paid attention in chemistry class and, consequently, they do not know that when the temperature drops so does the pressure. Luckily you’ve been hired to shed light on the case. You recall the ideal gas law as: $PV = nRT$, where $T$ is the temperature in Kelvin. The following method already exists:

```java
/**
 * @param degreesFahrenheit
 * @return kelvin
 */
public static double fahrenheitToKelvin(double degreesFahrenheit) {
    return (degreesFahrenheit + 459.67) * (5.0 / 9.0);
}
```

Write `calculatePressure` using the above method to convert the temperature from Fahrenheit to Kelvin:

```java
/**
 * @param v the volume of the gas
 * @param n the amount of substance of gas (a.k.a the number of moles)
 * @param r the ideal, or universal, gas constant
 * @param tF the temperature of the gas in Fahrenheit
 * @return the pressure of the gas
 */
public static double calculatePressure(double v, double n, double r, double tF) {
    double kelvin = fahrenheitToKelvin(tF);
    return (v * n * r) / kelvin;
}
```
In this problem you develop a recursive solution to draw this image.

(1) (2 points) Describe the substructure of the image you see above.

(2) (2 points) Describe the base case of the recursion present in the above image.
(3) (10 points) Complete `recursiveCarpet` to draw (something reasonably resembling) the image. Note that the center square is \( \frac{1}{3} \) the width of the window. Do not overly stress about the exact values to produce this image. Reasonable values will be accepted.

The `filledSquare` method from the StdDraw API can be used to draw a filled square. Its signature is:

```java
public static void filledSquare(double x, double y, double halfLength)
```

```java
/**
 * @param xCtr x-coordinate of the center of this carpet
 * @param yCtr y-coordinate of the center of this carpet
 * @param halfLength half of the length of this carpet
 */
public static void recursiveCarpet(double xCtr, double yCtr, double halfLength) {
```

```java
public static void main(String[] args) {
    double halfCarpetLength = 0.5;
    StdDraw.setXscale(-halfCarpetLength, halfCarpetLength);
    StdDraw.setYscale(-halfCarpetLength, halfCarpetLength);
    recursiveCarpet(0, 0, halfCarpetLength);
}
```
12. A palindrome is defined to be a sequence of characters that reads the same backward as forward.

For example, the sometimes attributed to Napoleon quote "able was i ere i saw elba" is a palindrome. The words "mom" and "dad" are also palindromes.

"cytron" is NOT a palindrome as its reverse is "nortyc". "bonbon" is NOT a palindrome since "bonbon" reversed is "nobnob".

You will write two different methods `isPalindromeViaLoop` and `isPalindromeViaRecursion`, each of which will check to see if a specified String is a palindrome. You may use provided String utility methods: `onlyFirst`, `onlyLast`, `allButFirstAndLast`, and `length`. If you are interested, implementations for these utility methods are included on the final page as a reference.

For further clarification, we have provided two example mains, each of which calls its own version of the palindrome check. Each main produces the exact same output which is listed below (once).

**main which calls isPalindromeViaRecursion:**

```
// Array of phrases to test
String[] texts = { "able was i ere i saw elba", "abba", "mom", "dad", "a", "ab", "bonbon", "cytron" };

// Loop to test isPalindromeViaRecursion()
for (int i=0; i<texts.length; ++i) {
    System.out.println("result: "+isPalindromeViaRecursion(texts[i]) + "text: "+texts[i]);
}

// Loop to test isPalindromeViaLoop()
for (int i=0; i<texts.length; ++i) {
    System.out.println("result: "+isPalindromeViaLoop(texts[i]) + "text: " +texts[i]);
}
```

The following table would be printed by each loop (it’d be printed twice):

<table>
<thead>
<tr>
<th>result:</th>
<th>text:</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>able was i ere i saw elba</td>
</tr>
<tr>
<td>true</td>
<td>abba</td>
</tr>
<tr>
<td>true</td>
<td>mom</td>
</tr>
<tr>
<td>true</td>
<td>dad</td>
</tr>
<tr>
<td>true</td>
<td>a</td>
</tr>
<tr>
<td>false</td>
<td>ab</td>
</tr>
<tr>
<td>false</td>
<td>bonbon</td>
</tr>
<tr>
<td>false</td>
<td>cytron</td>
</tr>
</tbody>
</table>
(1) (4 points) For the String “able was i ere i saw elba”:
   i. Draw a box around the next recursive sub-problem (that is: what String would be passed to
      the next recursive call):
      
      able was i ere i saw elba

   ii. Draw a box around the base case:
      
      able was i ere i saw elba

(2) (8 points) Write code to detect if a string is a palindrome via a recursive implementation. You
    may use these String utilities (implementations provided on final page of exam):

    private static String onlyFirst(String input)
    private static String onlyLast(String input)
    private static String allButFirstAndLast(String input)
    private static int length(String input)

    You must produce a recursive solution here.
    NO CREDIT will be awarded for a loop based solution.

    public static boolean isPalindromeViaRecursion(String s) {

    }
(3) (8 points) Write code to detect if a string is a palindrome via a loop-based, iterative implementation. You may use these String utilities (implementations provided on final page of exam):

```java
private static String onlyFirst(String input)
private static String onlyLast(String input)
private static String allButFirstAndLast(String input)
private static int length(String input)
```

To repeat: you must produce a loop-based solution on this page.
NO CREDIT will be awarded for a recursive solution or non-looping approach here.

```java
public static boolean isPalindromeViaLoop(String s) {

...........................
...........................
...........................
...........................
...........................
...........................
...........................
...........................
...........................
...........................
...........................
...........................
...........................
...........................
...........................

}
```
String method implementations provided as a reference.

```java
/**
 * @param input the specified String
 * @param b the index of the (inclusive) beginning of the slice
 * @param e the index of the (exclusive) end of the desired slice
 * @return a new String which is a slice of the input String from [b, e).
 */
private static String slice(String input, int b, int e) {
    String s = "";
    for (int i = b; i < e; ++i) {
        s = s + input.charAt(i);
    }
    return s;
}

/**
 * @param input the specified String
 * @return a new String which is the first character of input
 */
private static String onlyFirst(String input) {
    return slice(input, 0, 1);
}

/**
 * @param input the specified String
 * @return a new String which is the last character of input
 */
private static String onlyLast(String input) {
    return slice(input, input.length() - 1, input.length());
}

/**
 * @param input the specified String
 * @return a new String which the characters of input with the first and last character removed
 */
private static String allButFirstAndLast(String input) {
    return slice(input, 1, input.length() - 1);
}

/**
 * @param input the specified String
 * @return the length of input
 */
private static int length(String input) {
    return input.length();
}
```