Computer Science 300

# Example Exam

October 28, 2025; 90 minutes
100 points (10% of final grade)
Instructors: Hobbes LeGault and Jim Williams



**Your Name:** _____

**Wisc Email:** _____ @wisc.edu

**SCANTRON: Fill in these fields (left to right) on the scantron form using a #2 pencil:**

1. LAST NAME (family name) and FIRST NAME, as much as there is space to enter
2. IDENTIFICATION NUMBER as your campus ID number (from your Wiscard)
3. **SPECIAL CODES ABC**: enter the last three digits of the exam ID number at the top of this page
4. **SPECIAL CODE F**: enter the number **1**, indicating that you have exam version 1

## This exam contains 3 parts and is worth a total of 100 points:

**Part 1** contains 10 Simple Choice questions worth 2 points each, for a total of **20** points possible.
**Part 2** contains 10 Multiple Choice questions worth 3 points each, for a total of **30** points possible.

Fill in ONE (1) ***best*** answer bubble on the scantron for each of these questions.

**Part 3** contains 2 written questions, for a total of **50** points possible.

ONLY answers written directly on the exam booklet will be considered for part 3;
answers written on scratch paper will not be graded.

---

## Appropriate Academic Conduct for Exams

- **Keep your answers covered** so they cannot be viewed by other students during the exam.
- The only references you may use are contained within ***this*** exam booklet. You may not use any electronic devices or other students' exam materials during this exam.
- **Do not take photos** or otherwise record the contents of this exam at any time.

Any violation of this code of conduct will result in a **zero** on this exam, and other disciplinary actions as appropriate in accordance with the policies of the Office of Student Conduct and Community Standards at the University of Wisconsin–Madison.

Disclaimer: the following are provided for your reference only, and inclusion of information here does not guarantee its use in an exam question.

## Operator Precedence Table:

| Level | Operator | Description | Associativity |
|---|---|---|---|
| higher | `( <expression> )` | grouping with parentheses | left to right |
| | `[ ] ( ) .` | array index, method call, member access (dot operator) | left to right |
| | `++ --` | post-increment, post-decrement | left to right |
| | `++ -- + - !` | pre-increment, unary plus/minus, logical negation | right to left |
| | `(type)` | casting | right to left |
| | `* / %` | multiplication, division, modulus | left to right |
| | `+ - +` | addition, subtraction, concatenation | left to right |
| | `< <= > >= instanceof` | relational and Java' s instanceof operator | left to right |
| | `== !=` | equality | left to right |
| | `&&` | conditional AND (short-circuits) | left to right |
| | `\|\|` | conditional OR (short-circuits) | left to right |
| | `? :` | ternary conditional | right to left |
| lower | `= += -= *= /= %=` | assignment | right to left |

## Methods from the `java.lang.Object` class: (superclass of all classes in Java)

| | |
|---|---|
| `String toString()` | Returns a `String` representation of the object. This is the hash code of the instance unless `toString()` has been overridden. |
| `boolean equals(Object o)` | Returns `true` if the object referenced as `o` is the same as `this`. It is often overridden (redefined) by instantiable classes. |

## Methods from the `java.lang.Integer` class: (which implements `Comparable`)

| | |
|---|---|
| `static int parseInt(String s) throws NumberFormatException` | Converts `s` into the corresponding `int` value. Throws `NumberFormatException` when `s` can't be converted. |
| `int intValue()` | Return the `int` value of *this* `Integer` instance. |
| `int compareTo(Integer n)` | Returns a negative value if this `Integer` is smaller than `n`, zero if they are equal, otherwise a positive value. |
| `Integer(int n)` | Constructs a new `Integer` object representing `n`. |

## Methods from the `java.lang.Double` class: (which implements `Comparable`)

| | |
|---|---|
| `Double(double d)` | Constructs a new `Double` object representing `d`. |

## The `java.lang.Comparable<T>` interface:

| | |
|---|---|
| `int compareTo(T obj)` | Returns a negative value if `this` is less than `obj`, zero if they are equal, and a positive value if `this` is greater than `obj`. |

## Methods from the `java.util.Scanner` class:

| | |
|---|---|
| `Scanner(String s)` | Creates a `Scanner` to read the `String s`. |
| `Scanner(File fn) throws` | Create a `Scanner` to read from a file `fn`. |

| | |
|---|---|
| *FileNotFoundException* | |
| `void close()` *throws IOException* | Closes the stream and any associated file. |
| `boolean hasNextLine()` | Returns `true` if there is another line of input. |
| `String next()` *throws NoSuchElementException* | Returns the next word only, as a `String`. |
| `int nextInt()` *throws NoSuchElementException* | Returns the next word only, as an `int`. |
| `String nextLine()` *throws NoSuchElementException* | Returns the next line as a `String`. |

## Methods from the `java.lang.String` class: (which implements `Comparable`)

| | |
|---|---|
| `int length()` | Returns number of characters in the `String` |
| `char charAt(int index)` | Returns character at the specified `index` of the `String` |
| `String substring(int beginIndex, int endIndex)` | Returns a *new* `String` that is a substring of this `String`. The substring begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`. |
| `String toLowerCase()` | Returns a *new* `String` that is the *lowercase* version of this string. |
| `int indexOf(String s)` | Returns the index within this string of the first occurrence of the specified substring, or `-1` if not found. |
| `static String valueOf(Object obj)` | Returns `null` when `obj` is `null`, otherwise returns `obj.toString()`. |
| `boolean equals(String s)` | Returns `true` if the contents of this `String` are the same as the contents of String `s`. |
| `int compareTo(String s)` | Returns a negative value if this `String` is alphabetically earlier than `s`, zero if they are equal, and a positive value if this `String` is alphabetically later than `s`. |
| `String concat(String s)` | Returns a new `String` containing `s` concatenated to the end of this `String`. |

## Methods from the `java.util.Arrays` class:

| | |
|---|---|
| `static String toString(E[] array)` | Returns a `String` representation of any type (`E[]`) array. |
| `static void sort(E[] array)` | Sorts the specified array in memory. Type `E` must be `Comparable` or `Comparable<E>`. |

## Methods from the `java.util.ArrayList<E>` class: (*REMEMBER 0-based indexing)

| | |
|---|---|
| `ArrayList<E>()` | Constructs an empty list. |
| `ArrayList<E>(int cap)` | Constructs an empty list with initial capacity `cap`. |
| `int size()` | Returns the number of used elements in this list. |
| `E get(int index)` | Returns the item at the specified index in this list. throws `IndexOutOfBoundsException` if invalid index |
| `void add(E item)` | Adds the specified item to the end of this list. |
| `void add(int index, E item)` | Adds the specified item by inserting it into this list at the specified index. |
| `boolean addAll(Collection<E> c)` | Appends all of the elements in the specified collection to the end of this list, in the order they appear in `c`. |
| `E remove(int index)` *throws IndexOutOfBoundsException* | Removes and returns the item from the specified index; or throws exception when no element is at that index. |
| `boolean addAll(ArrayList<E> a)` | Appends all of the elements in the specified list to the end of this list, in the order they appear in the provided list. |
| `String toString()` | Calls `toString()` on each element in the list & returns a single comma-separated String of these results. |

## Exception Class Inheritance Hierarchy

```
public class Throwable extends Object
   public class Exception extends Throwable
      public class RuntimeException extends Exception
         public class ArithmeticException extends RuntimeException
         public class IndexOutOfBoundsException extends RuntimeException
            public class ArrayIndexOutOfBoundsException extends IndexOutOfBoundsException
         public class IllegalStateException extends RuntimeException
         public class IllegalArgumentException extends RuntimeException
            public class NumberFormatException extends IllegalArgumentException
         public class NullPointerException extends RuntimeException
      public class IOException extends Exception
         public class FileNotFoundException extends IOException
         public class EOFException extends IOException
```

## Typical Uses of Standard Exceptions

| | |
|---|---|
| `ArithmeticException` | Overflow or integer division by zero. |
| `ClassCastException` | Casting to a subclass of which this is not an instance |
| `NumberFormatException` | Illegal conversion of String to numeric type. |
| `IndexOutOfBoundsException` | Illegal index into an array or a String. |
| `ArrayIndexOutOfBoundsException` | Array accessed with an illegal index |
| `NullPointerException` | Illegal attempt to use a null reference. |
| `IOException` | Includes most I/O exceptions. |
| `FileNotFoundException` | File not found to open, or create. |
| `IllegalArgumentException` | Illegal argument passed to a method. |
| `IllegalStateException` | Method called when it should not have been. |

## Part I: Simple Choice (Questions 1-10, 2 points each)

1. A subclass constructor must always call the superclass constructor (either explicitly or implicitly).

   A. True
   B. False

2. What happens when display() is called?

```
class A {
    protected int x = 5;
}

class B extends A {
    public void display() {
        System.out.println(x);
    }
}
```

   A. Prints 5
   B. Compilation error - cannot access x

3. Which fix is required for this code to compile?

```
public void readFile() {
    FileReader fr = new FileReader("data.txt");
}
```

   A. Add throws IOException to the method signature
   B. No fix needed - FileNotFoundException is unchecked

The exam continues on the next page.

# Part II: Multiple Choice (Questions 11-20, 3 points each)

4. What is the output?

```
class Animal {
   public void sound() {
      System.out.println("Some sound");
   }
}
class Dog extends Animal {
   public void sound() {
      System.out.println("Bark");
   }
}
public class Main {
   public static void main(String[] args) {
      Animal a = new Dog();
      a.sound();
   }
}
```

A. Some sound
B. Bark
C. Compilation error
D. Runtime error

5. What happens?

```
interface Flyable {
    void fly();
}
class Bird implements Flyable {
    public void fly() { System.out.println("Flying"); }
    public void chirp() { System.out.println("Chirping"); }
}
public class Main {
   public static void main(String[] args) {
      Flyable f = new Bird();
      ((Bird) f).chirp();
   }
}
```

A. Prints "Chirping"
B. Prints "Flying"
C. Compilation error
D. Runtime ClassCastException

# Part III: Short Programming Section (Question 21, 50 points)

**All code for these questions *must* be written directly on pages 9 and 10**; any **code written on scratch** paper or any other page of the exam booklet ***will NOT be graded***. Feel free to design your solution on scratch paper first and then copy the final solution to the exam booklet.

Your solutions for any coding questions must be written in as close to **valid Java code** as possible. Pseudocode or English explanations will NOT be accepted for credit, but grading will focus on algorithmic correctness rather than syntax.

---

**Write your name and wisc email at the top of page 9.**
Failure to do so may delay your exam being properly graded.

---

21. This **recursive** method sums the elements in an oversize array:

- If there are no elements the sum is 0.0.
- Otherwise, sum all the elements.

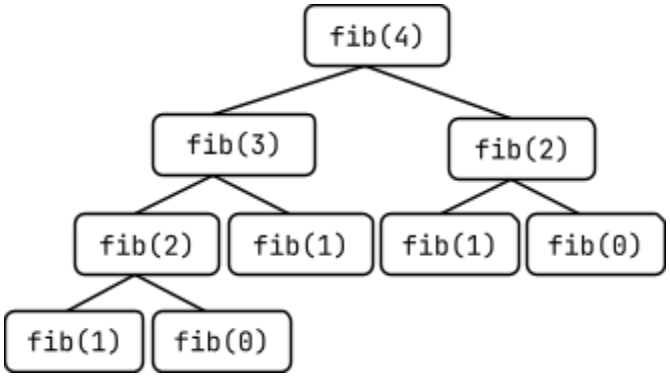
First, you must draw the ***recursion tree*** for this method using the array `[ 5, 1, 7, 4, 10 ]` and `size = 4`; then you will ***implement*** the recursive method.

---

**NO CREDIT WILL BE GRANTED**
**IF YOUR IMPLEMENTATION USES ITERATION (LOOPS).**

---

Recall that a recursion tree tracks the recursive CALLS and how progress is made toward the base case; you can use this later to build your full recursive solution. For example, the recursion tree for the Fibonacci sequence using an initial argument of 4 would look like this:



This page is intentionally left blank and may be used for scratch paper.



Any code written on this page will NOT be graded.

Name: _____

Wisc Email: _____ @ wisc.edu

## Part III Short Programming Answer (Question 21, 50 points)

1. Fill in the following blank to complete the method signature with the appropriate return type, according to the description on page 7.

```
public static double sum(int[] nums, int size) { ... }
```

2. Given this method signature and the description of the recursive function, draw the recursion tree starting with the call **`sum( [ 5, 1, 7, 4, 10 ], 4)`**. For full credit, show ALL ***recursive calls*** and ***arguments*** from the initial call at the TOP of the drawing to the base case at the BOTTOM. You do NOT need to include return values in this diagram.

3. Now, implement the method in the space provided below:

{

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

}

**Note**: you may not need to use all of the provided lines.



---

Double-check that you have answered all 20 questions on your SCANTRON bubble sheet, in addition to all blanks in this short answer question. When done, turn in your SCANTRON sheet along with your exam questions. Please have your UW ID ready.

---