1.1

What are the two parts of an ADT? Which part is accessible to a user and which is not? Explain the relationships between an ADT and a class; between an ADT and an interface; and between an interface and classes that implement the interface.

An ADT consists of data elements and methods that operate on that data. The user can access the operations, but cannot access the internal structure of the data elements. A class provides one way to implement an ADT in Java. A Java interface is a way to specify or describe an ADT. An interface defines a set of methods, and a class that implements an interface must implement these methods and define any necessary data fields.

1.2

Assume there is an interface named `Comparable` with the following definition:

```
public interface Comparable {
    int compareTo(Object obj);
}
```

Do you think class `String` implements interface `Comparable`? Provide a reason for your answer.

The `String` class implements the interface `Comparable` because in defines the method `compareTo`.

1.3

Correct each of the following statements that is incorrect, assuming that class `PDGUI` and class `PDConsoleUI` implement interface `PDUserInterface`.

a. `PDGUI p1 = new PDConsoleUI();`
b. `PDGUI p2 = new PDUserInterface();`
c. `PDUserInterface p3 = new PDUserInterface();`
d. `PDUserInterface p4 = new PDConsoleUI();`
e. `PDGUI p5 = new PDUserInterface();`
   `PDUserInterface p6 = p5;`
f. `PDUserInterface p7;`
   `p7 = new PDConsoleUI();`

a. `PDGUI p1 = new PDConsoleUI();`
   These are not the same class, even though they implement the same interface.
   `PDIGUI p1 = new PDIGUI();`
   or
   `PDConsoleUI p1 = PDConsoleUI()`
b. `PDGUI p2 = new PDUserInterface();`
   `PDIGUI` is a `PDUserInterface` not the other way around.
   `PDUserInterface p2 = new PDGUI();`
   is more appropriate.
c. `PDUserInterface p3 = new PDUserInterface();`
   You cannot instantiate an interface.

d. `PDUserInterface p4 = new PDConsoleUI();`
   This is a valid statement
e. `PDGUI p5 = new PDUserInterface();`
   `PDUserInterface p6 = p5;`
   The first statement is not valid (see b above). If the first statement read
   `PDUserniterface p5 = new PDGUI();`
   then the second statement would be valid and p6 and p5 would refer to the same object in memory.
f. `PDUserInterface p7;`
   `p7 = new PDConsoleUI();`
   These statements are valid.

1.4

Explain how an interface is like a contract.

An interface specifies methods that a class implementing that interface must provide. For each method specified, the name, return type, and the parameters are specified. For each parameter the type is specified. A user of a class implementing the interface can be assured that the methods are as specified and the developer of class must provide the specified methods. Thus, the interface defines a contract between the user of a class and its developer.

1.5

What are two different uses of the term *interface* in programming?

The way in which a user interacts with the program is known as the user interface. A set of methods that a class must provide and an optional set of constants is interface for a set of classes, and is defined using the Java key-word **`interface`**.

2.1

Explain the effect of each valid statement in the following fragment. Indicate any invalid statements.

```
Computer c1 = new Computer();
Computer c2 = new Computer("Ace", "AMD", 1.0, 160, 2.0);
Notebook c3 = new Notebook("Ace", "AMD", 2.0, 160, 1.8);
Notebook c4 = new Notebook("Bravo", "Intel", 1.0, 160, 15.5, 7.5, 2.0);
System.out.println(c2.manufacturer + "," + c4.processor);
System.out.println(c2.getDiskSize() + "," + c4.getRamSize());
System.out.println(c2.toString() + "\n" + c4.toString());
```

`Computer c1 = new Computer();`
Not valid: Computer does not have a no-argument constructor
`Computer c2 = new Computer("Ace", "AMD", 1.0, 160, 2.0);`
Valid: A new Computer object is created with a manufacture Ace, a processor AMD, 1.0 gigabytes of ram, 160 gigabytes of disk, and a processor speed of 2.0 GHz.
`Notebook c3 = new Notebook("Ace", "AMD", 2.0, 160, 1.8);`
Not valid: The parameters to define the screen size and weight are missing.
`Notebook c4 = new Notebook("Bravo", "Intel", 1.0, 160, 2.0, 15.5, 7.5);`
Valid: A new Notebook object is created with a manufacturer Bravo, a processor Intel, 1.0 gigabytes of ram, 160 gigabytes of disk, a processor speed of 1.8 GHz, a screen size of 15.5 and a weight of 7.5.
`System.out.println(c2.manufacturer + "," + c4.processor);`
Not valid: `manufacturer` and `processor` are **`private`** members of `Computer` and `Notebook`.

```
System.out.println(c2.getDiskSize() + "," + c4.getRamSize());
```
Valid: outputs the string `160, 1.0`
```
System.out.println(c2.toString() + "\n" + c4.toString());
```
Valid outputs the following:
```
Manufacturer: Ace
CPU: AMD
RAM: 1.0 megabytes
Disk: 160 gigabytes
Processor speed: 2.0 gigahertz
Manufacturer: Bravo
CPU: Intel
RAM: 1.0 megabytes
Disk: 160 gigabytes
Processor speed: 15.5 gigahertz
```

2.2

Indicate where in the hierarchy you might want to add data fields for the following and the kind of data field you would add.

Cost
The battery identification
Time before battery discharges
Number of expansion slots
Wireless Internet available

Cost is common to computers and notebooks, so it should be defined in `Computer` of type **`double`**.

The battery identification is unique to notebooks, so it should be defined in `Notebook` of type `String`.

Time before battery discharges is unique to notebooks, so it should be defined in `Notebook` of type **`double`**.

Wireless Internet available is common to computers and notebooks, so it should be defined in `Computer` of type **`boolean`**.

2.3

Can you add the following constructor to class `Notebook`? If so, what would you need to do to class `Computer`?

```
public Notebook() {}
```

Yes, if you provided you defined a no-argument constructor for class `Computer`.

3.1

Explain the effect of each of the following statements. Which one(s) would you find in class `Computer`? Which one(s) would you find in class `Notebook`?

```
super(man, proc, ram, disk, procSpeed);
this(man, proc, ram, disk, procSpeed);
```

```
super(man, proc, ram, disk, procSpeed);
```

This statement calls the constructor of the superclass (Computer) with the parameter types `String`, `String`, `int`, `int`, `double`. This statement must the first statement of a constructor in the class `Notebook`.

`this(man, proc, ram, disk, procSpeed);`
This statement calls the constructor in the class Computer with the parameter types `String`, `String`, `int`, `int`, `double`. This statement must be the first statement of a constructor in the class `Computer`.

3.2

Indicate whether methods with each of the following signatures and return types (if any) would be allowed and in what classes they would be allowed. Explain your answers.

```
Computer()
Notebook()
int toString()
double getRamSize()
String getRamSize()
String getRamSize(String)
String getProcessor()
double getScreenSize()
```

`Computer()`
Allowed in class Computer. It is a no-argument constructor.
`Notebook()`
Allowed in class Notebook. It is a no-argument constructor.
`int toString()`
Not valid. The return type of method `toString` must be String.
`double getRamSize()`
Valid, and would be in class `Computer`. Could also be in class `Notebook`.
`String getRamSize()`
This would be valid, if the previous method was not defined. You can only have one method with the name `getRamSize()`.
`String getRamSize(String)`
This would be valid if the `String` parameter were given a name, and it could be defined in either `Computer` or `Notebook`. The `String` parameter could be ignored, but providing it allows for an overloading of the method `toString`.
`String getProcessor()`
Valid and would be defined in the class `Computer`.
`double getScreenSize()`
Valid and would be defined in the class `Notebook`.

3.3

For the loop body in the following fragment, indicate which method is invoked for each value of `i`. What is printed?

```
Computer comp[] = new Computer[3];
comp[0] = new Computer("Ace", "AMD", 3, 160, 2.4);
comp[1] = new Notebook("Dell", "Intel", 4, 350, 2.2, 15.5, 7.5);
comp[2] = comp[1];
for (int i = 0; i < comp.length; i++) {
    System.out.println(comp[i].getRamSize() +"\n" +
                       comp[i].toString());
}
```

For `i = 0`, `Computer.toString` is invoked and

```
Manufacturer: Ace
CPU: AMD
RAM: 3.0 gigabytes
Disk: 160 gigabytes
```

```
Processor speed: 2.4 gigahertz
```
is printed

For i = 1, `Notebook.toString` is invoked and
```
Manufacturer: Dell
CPU: Intel
RAM: 4.0 gigabytes
Disk: 350 gigabytes
Processor speed: 2.2 gigahertz
```
is printed

For i = 2, `Notebook.toString` is invoked and
```
Manufacturer: Dell
CPU: Intel
RAM: 4.0 gigabytes
Disk: 350 gigabytes
Processor speed: 2.2 gigahertz
```
is printed

3.4

When does Java determine which `toString` method to execute for each value of `i` in the **`for`** statement in the preceding question: at compile time or at run time? Explain your answer.

The determination is made at run-time. The actual type of the reference stored in each element of the array determines which overloaded method is called.

4.1

What are two important differences between an abstract class and an actual class? What are the similarities?

An abstract class can contain declarations of abstract methods and an abstract class cannot be instantiated. Both abstract and actual classes can contain method definitions, and you can declare a variable that is of an abstract class type or of an actual class type.

4.2

What do abstract classes and interfaces have in common? How do they differ?

Both can declare abstract methods, and you can declare a variable that is of an interface type or of an abstract class type. Both can define static constants. Neither can be instantiated. Interfaces cannot contain method definitions. A class may implement multiple interfaces, but extend only one class. And an interface may extend more than one interface.

4.3

Explain the effect of each statement in the following fragment and trace the loop execution for each value of `i`, indicating which `doubleValue` method executes, if any. What is the final value of `x`?

```
Number[] nums = new Number[5];
nums[0] = new Integer(35);
nums[1] = new Double(3.45);
nums[4] = new Double("2.45e6");
double x = 0;
for (int i = 0; i < nums.length; i++) {
    if (nums[i] != null)
        x += nums[i].doubleValue();
}
```

`Number[] nums = new Number[5];`
Declares an array `nums` of type `Number` and initializes it to five **`null`** values.
`nums[0] = new Integer(35);`
The entry with index 0 refers to an `Integer` with the value of 35.
`nums[1] = new Double(3.45);`
The entry with index 1 refers to a `Double` with the value of 3.45.
`nums[4] = new Double("2.45e6");`
The entry with index 4 refers to a `Double` with the value of 2,450,000.
`double x = 0;`
Declares the variable `x` of type **`double`** and initializes it to 0.
`for (int i = 0; i < nums.length; i++) {`
Initiates a loop with the index `i` that will take on the values 0, 1, 2, 3, 4
`    if (nums[i] != null)`
Tests to see if the value at index `i` of the array `nums` is not **`null`**, if so, the next statement is executed
`        x += nums[i].doubleValue();`
Adds the value at index `i` to the variable `x`.

The final value of x is 2,450,038.45.

4.4

What is the purpose of the **`if`** statement in the loop in Question 3? What would happen if it were omitted?

The **`if`** statement ensures that the array entry contains a valid reference. If were omitted a `NullPointerException` would be thrown when `i` was 3 since `nums[3]` was not initialized to a value.

5.1

Indicate the effect of each of the following statements:

```
Object o = new String("Hello");
String s = o;
Object p = 25;
int k = p;
Number n = k;
```

`Object o = new String("Hello");`
Declares the variable o and initializes it to reference the String "Hello".
`String s = o;`
Not a valid statement. The variable o is not of type String.
`Object p = 25;`
Not a valid statement. The constant 25 is not a class type.
`int k = p;`
Not a valid statement. The variable p is not an int.

```
Number n = k;
```
Not a valid statement. The variable k is not of type Number of one of its subtypes.

5.2

Rewrite the invalid statements in Question 1 to remove the errors.

```
Object o = new String("Hello");
String s = (String) o;
Object p = new Integer(25);
int k = (Integer) p;
Number n = new Integer(k);
```

6.1

Explain the key difference between checked and unchecked exceptions. Give an example of each kind of exception. What criterion does Java use to decide whether an exception is checked or unchecked?

6.2

What is the difference between the kind of unchecked exceptions in class `Error` and the kind in class `RuntimeException`?

Exceptions of type `Error` represent conditions that are generally unrecoverable, e.g., `OutOfMemoryError`. In general, the only option is to terminate the program, and thus `Error`s should not be caught. Exceptions of type `RuntimeException` are generally caused by programmer error, e.g., `NullPointerException`. `RuntimeException`s can be caught and perhaps recovered from, or at least meaningful error messages issued.

6.3

List four subclasses of `RuntimeException`.

`NullPointerException`, `IndexOutOfBoundsException`, `ArithmeticException`, and `ClassCastException`.

6.4

List two subclasses of `IOException`.

`FileNotFoundException` and `EOFException`

6.5

What happens in the `main` method preceding the exercises if an exception of a different type occurs in method `processPositiveInteger`?

The JVM will process an uncaught exception. A stack trace will be output to `System.err` and the program execution will terminate.

6.6

Trace the execution of method `getIntValue` if the following data items are entered by a careless user. What would be displayed?

```
ace
7.5
-5
```

```
    public static int getIntValue(Scanner scan) {
```
The method is entered and the scan parameter is bound to its actual argument
```
        int nextInt = 0;          // next int value
```
`nextInt` is set to 0
```
        boolean validInt = false; // flag for valid input
```
`validInt` is set to false
```
        while (!validInt) {
```
The loop body is entered
```
            try {
                System.out.println("Enter number of kids:");
```
`Enter number of kids:` is written to `System.out`
```
                nextInt = scan.nextInt();
```
`ace` in ecountered as the next input

`InputMismatchException` is thrown
```
            } catch (InputMismatchException ex) {
```
The exception object is bound to the parameter `ex`
```
                scan.nextLine();  // clear buffer
```
The current input line (containing `ace`) is discarded
```
                System.out.println("Bad data -- enter an integer:");
```
`Bad data -- enter an integer:` is written to `System.out`
```
            }
```
Control goes to the `while` statement
```
        while (!validInt) {
```
The loop body is entered
```
            try {
                System.out.println("Enter number of kids:");
```
`Enter number of kids:` is written to `System.out`
```
                nextInt = scan.nextInt();
```
`7.5` in ecountered as the next input

`InputMismatchException` is thrown
```
            } catch (InputMismatchException ex) {
```
The exception object is bound to the parameter `ex`
```
                scan.nextLine();  // clear buffer
```
The current input line (containing `7.5`) is discarded
```
                System.out.println("Bad data -- enter an integer:");
```
`Bad data -- enter an integer:` is written to `System.out`
```
            }
```
Control goes to the `while` statement
```
        while (!validInt) {
```

The loop body is entered

```
            try {
                System.out.println("Enter number of kids:");
```

Enter number of kids: is written to System.out

```
                nextInt = scan.nextInt();
```

-5 in encountered as the next input

nextInt is set to -5

```
                validInt = true;
```

validInt is set true.

```
            }
         }
```

Control goes to the while statement

```
        while (!validInt) {
```

The loop body is skipped

```
        }
        return nextInt;
```

The value -5 is returned to the caller.

Output is as follows:

```
Enter number of kids:
ace
Bad data -- enter an integsr:
Enter number of kids:
7.5
Bad data -- enter an integsr:
Enter number of kids:
-5
```

6.7

Trace the execution of method `main` preceding the exercises if the data items in Question 6 were entered. What would be displayed?

```
    public static void main(String[] args) {
```

The main method is entered and the command line arguments are bound to the array args

```
        Scanner scan = new Scanner (System.in);
```

A Scanner object scan is created and bound to System.in

```
        try {
            int num = getIntValue(scan);
```

See the answer to question 6.6.

num is set to -5

```
            processPositiveInteger(num);
    public static void processPositiveInteger(int n) {
```

The method processPositiveInteger is called with the parameter n bound to the value -5.

```
        if (n < 0)
```

Control passes to the next statement

```
            throw new IllegalArgumentException(
                    "Invalid negative argument");
```

An IllegalArgumentException is created with the message Invalid negative argument and it is thrown

```
        } catch (IllegalArgumentException ex) {
```

The `IllegalArgumentException` object is bound to the parameter ex

```
            System.err.println(ex.getMessage());
```

The string Invalid negative argument is written to System.err.

```
            System.exit(1); // error indication
```

The JVM terminates program execution.

The following is the output:

```
Enter number of kids:
ace
Bad data -- enter an integsr:
Enter number of kids:
7.5
Bad data -- enter an integsr:
Enter number of kids:
-5
Invalid negative argument
```

7.1

Consider the following declarations:

```
package pack1;
public class Class1 {
private int v1;
protected int v2;
int v3;
public int v4;
}
package pack1;
public class Class2 {...}
package pack2;
public class Class3 extends pack1.Class1 {...}
package pack2;
public class Class4 {...}
```

a. What visibility must variables declared in `pack1.Class1` have in order to be visible in `pack1.Class2`?
b. What visibility must variables declared in `pack1.Class1` have in order to be visible in `pack2.Class3`?
c. What visibility must variables declared in `pack1.Class1` have in order to be visible in `pack2.Class4`?

a. What visibility must variables declared in `pack1.Class1` have in order to be visible in `pack1.Class2`?
The public (`v4`), protected (`v2`), and default (`v3`) variables of `pack1.Class1` are visible in `pack1.Class2`
b. What visibility must variables declared in `pack1.Class1` have in order to be visible in `pack2.Class3`?
The public (`v4`) and protected (`v2`) variables of `pack1Class1` are visible in `pack2.Class3`
c. What visibility must variables declared in `pack1.Class1` have in order to be visible in `pack2.Class4`?
Only the public (v4) variables declared in `pack1.Class1` are visible in `pack2.Class4`.

8.1

Explain why `Shape` cannot be an actual class.

Because the methods `computeArea`, `computePerimeter`, and `readShapeData` all depend on the actual Shape class, and there is no general way to define them.

8.2

Explain why `Shape` cannot be an interface.

The `Shape` class has the data field `shapeName` and the actual method `getShapeName`. Interfaces cannot contain data fields or actual methods.