

1.1.1

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.

1.1.3

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
`PDUserInterface 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.

1.1.5

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**.

1.2.1

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

1.2.3

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.

1.3.1

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 be 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.

1.3.3

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

1.4.1

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.

1.4.3

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.

1.5.1

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` or one of its subtypes.

1.6.1

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?

1.6.3

List four subclasses of `RuntimeException`.

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

1.6.5

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.

1.6.7

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 integer:

Enter number of kids:

7.5

Bad data -- enter an integer:

Enter number of kids:

-5

Invalid negative argument

1.7.1

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 {...}
```

- What visibility must variables declared in `pack1.Class1` have in order to be visible in `pack1.Class2`?
- What visibility must variables declared in `pack1.Class1` have in order to be visible in `pack2.Class3`?
- 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 `pack1.Class1` 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`.

1.8.1

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.