A.1.1

What is the Java Virtual Machine? Is it hardware or software? How does its role differ from that of the Java compiler?

The Java Virtual Machine (JVM) is software that simulates the execution of a hypothetical computer. The JVM's role differs from that of the Java Compiler in that the compiler is the device that translates the Java Source Code to the Java Byte Code, whereas the JVM actually takes the Java Byte Code and executes and interprets the commands and carries them out. Some versions of the JVM translate the Java Byte Code into native machine code at run-time. This is known as "just in time" compilation. The "hot spot" JVM that is included in the Java Development Kit from Sun observes the execution of the byte code as it is interpreted, and performs just in time compilation on code that it observes is repeatedly executed.

A.1.3

Explain the relationship between a class and an object. Which is general and which is specific?

A class is a named description for a group of entities that have the same characteristics. The class has properties and methods that can be performed on it. The object is a specific defined instance of a class. The class is more general; the object is more specific.

A.2.1

For the following assignment statement, assume that `x`, `y` are type **`double`** and `m`, `n` are type **`int`**. List the order in which the operations would be performed. Include any widening and narrowing conversions that would occur.

```
m = (int) (x * y + m / n / y * (m + x));
```

1. x is multiplied by y
2. m is divided by n
3. the result of step 2 (m/n) is widened to double
4. the result of step 4 is divided by y
5. m is widened to double and x is added
6. the result of step 5 is multiplied by the result of step 5
7. the result of step 6 is added to the result of step 1
8. the result of step 7 is narrowed to int (fraction part dropped) and stored into m

A.2.3

What is the difference between a reference variable and a primitive-type variable?

When a primitive data type is declared as a variable, the memory cell allocated for the variable is the actual data. When the variable is a reference variable to an object, the memory allocated for the variable is actually the address of the object referenced.

A.3.1

What is the purpose of the **break** statement in the preceding **switch** statement? List the statements that would execute when `operator` is `'-'` with the **break** statements in place and if they were removed.

*The **break** statement causes execution to leave the scope of the **switch** statement (closing }). If the break statements were removed and the `operator` was `'-'` then the following statements would be executed:*

```
result = x - y;
subtractOp++;
result = x * y;
multiplyOp++;
result = x / y;
divideOp++;
```

A.4.1

Identify the escape sequences in the following string. Show how this line would be displayed. Which of the escape sequences could be replaced by the second character of the pair without changing the effect?

```
System.out.println(
"Jane\'s motto is \n\"semper fi\"\n, according to Jim");
```

The escape sequences are:

`\'` representing `'`

`\n` representing a new-line

`\"` representing `"`

The string is displayed as follows:

```
Jane's motto is
"simper fi"
according to Jim
```

The `\'` could be replaced by `'` without changing the result.

A.5.1

Evaluate each of these expressions.

```
"happy".equals("Happy")
"happy".compareTo("Happy")
"happy".equalsIgnoreCase("Happy")
"happy".equals("happy".charAt(0) + "Happy".substring(1))
"happy" == "happy".charAt(0) + "Happy".substring(1)
```

`"happy".equals("Happy")`	*false*
`"happy".compareTo("Happy")`	*+1*
`"happy".equalsIgnoreCase("Happy")`	*true*

`"happy".equals("happy".charAt(0) + "Happy".substring(1))`	true
`"happy" == "happy".charAt(0) + "Happy".substring(1)`	false

A.5.3

Rewrite the following statements using `StringBuilder` objects:

```
String myName = "Elliot Koffman";
String myNameFirstLast = myName;
myName = myName.substring(7) + ", " + myName.substring(0, 6);
```

The first two statements do not use a StringBuilder. The last statement can be rewritten using a StringBuilder as follows:

```
StringBuilder temp = myName.substring(7);
temp.append(", ");
temp.append(myName.substring(0, 6);
myName = temp.toString();
```

A.5.5

Revise Exercise 4 to insert a newline character between the words in `result`.

```
StringBuilder result = new StringBuilder();
String sentence = "Let's all learn how to program in Java";
String[] tokens = sentence.split(\\s+);
for (String token : tokens) {
  result.append(token);
  result.append('\n');
}
```

A.6.1

Do you think objects of a wrapper type are immutable or not? Explain your answer.

The objects of the wrapper types are immutable because there are no methods defined that will change their value.

A.7.1

Explain why methods have public visibility but data fields have private visibility.

Methods implement the operations on the data type represented by the class and data fields represent the attributes or state of the object. Only the operations (methods) should be able to change the values of the attributes. Therefore, the data fields are declared private, and the methods are declared public. In this way users of the class can access the methods but cannot directly access the data fields.

A.7.3

Trace the execution of the following statements.

```
Person p1 = new Person("Adam", "Jones", "wxyz", 0);
p1.setBirthYear(1990);
```

```
Person p2 = new Person("abcd");
p2.setGivenName("Eve");
p2.setFamilyName(p1.getFamilyName());
p2.setBirthYear(p1.getBirthYear() + 10);
if (p1.equals(p2))
    System.out.println(p1 + "\nis same person as\n\n" + p2);
else
    System.out.println(p1 + "\nis not the same person as\n\n" + p2);
```

```
Person p1 = new Person("Adam", "Jones", "wxyz", 0);
```

Creates a Person object with the givenName Adam, the familyName Jones, an ID of wxyz and a birthYear of 0 and assigns a reference to it to the variable p1.

```
p1.setBirthYear(1990);
```

The birthYear of p1 (Adam Jones) is set to 1990

```
Person p2 = new Person("abcd");
```

Creates a Person object with all data fields set to null or 0 except for ID, which is set to abcd, is created and assigns a reference to it to the variable p2.

```
p2.setGivenName("Eve");
```

The givenName of the Person object referenced by p2 is set to "Eve"

```
p2.setFamilyName(p1.getFamilyName());
```

The familyName of the Person object referenced by p2 is set to "Jones"

```
p2.setBirthYear(p1.getBirthYear() + 10);
```

The birthYear of the Person object referenced by p2 is set to "2000"

```
if (p1.equals(p2))

    System.out.println(p1 + "\nis same person as\n\n" + p2);

else

    System.out.println(p1 + "\nis not the same person as\n\n" + p2);
```

Output is

```
Given name: Adam
Family name: Jones
ID number: wxyz
Year of birth: 1990

is not the same person as

Given name: Eve
Family name: Jones
ID number: abcd
```

```
Year of birth: 2000
```

A.8.1

Show the output that would be displayed by method main following Listing A.3.

```
Given name: Elliot
Family name: K
ID number: 123
Year of birth: 1942

Given name: Paul
Family name: W
ID number: 234
Year of birth: 1945
```

A.8.3

What is the output of the following sample code fragment?

```
int[] x;
int[] y;
int[] z;
x = new int[20];
x[10] = 0;
y = x;
x[10] = 5;
System.out.println(x[10] + ", " + y[10]);
x[10] = 15;
z = new int[x.length];
System.arraycopy(x, 0, z, 0, 20);
x[10] = 25;
System.out.println(x[10] + ", " + y[10]+ ", " + z[10]);
```

```
5 + 5
25 + 25 + 15
```

A.8.5

Assume there is no initializer list for the Pascal triangle and you are trying to build up its rows. If row i has been defined, write statements to create row `i + 1`.

```
pascal[i + 1] = new int[i + 2];
pascal[i + 1][0] = 1;
pascal[i + 1][i + 1] = 1;
for (int j = 1; j < i + 1; j++)
    pascal[i+1][j] = pascal[i][j-1] + pascal[i][j];
```

A.9.1

Show the statements that would be required, using Swing, to read and store the data for a `Person` object prior to calling the constructor with four parameters.

```
String givenName = JOptionPane.showInputDialog("Enter given name");
String familyName = JOptionPane.showInputDialog("Enter family name");
String id = JOptionPane.showInputDialog("Enter id");
String birthYear = JOptionPane.showInputDialog("Enter birth year");
Person p =
  new Person(givenName, familyName, id, Integer.parseInt(birtyYear));
```

A.10.1

Show the statements that would be required, using the console for input, to read and store the data for a `Person` object prior to calling the constructor with four parameters.

```
Scanner scan = new Scanner(System.in);
System.out.println("Enter given name");
String givenName = scan.next();
System.out.println("Enter family name");
String familyName = scan.next();
System.out.println("Enter id");
String id = scan.next();
System.out.println("Enter birth year");
String birthYear = scan.next();
Person p =
  new Person(givenName, familyName, id, Integer.parseInt(birtyYear));
```

A.10.3

What would happen if the output file name matched the name of a file already saved on disk? What could happen if the user forgets to close an output file?

If an output file name matches the name of a file already saved on a disk then the old file is deleted and the data written replaces it. If the user forgets to close an output file, some or all of the data may not be written to the destination.

A.11.1

Assume that method `main` calls method `first` at line 10 of class `MyApp`, method `first` calls method `second` at line 10 of class `Others`, and method `second` calls method `parseInt` at line 20 of class `Other`. These calls result in a `NumberFormatException` at line 430 of class `Integer`. Show the stack trace.

```
Exception in thread "main" java.lang.NumberFormatException
   at java.lang.Integer.parseInt(Integer.java 430)
   at Other.second(Other.java:20)
   at Others.first(Others.java:10)
   at MyApp.main(MyApp.java:10)
```

A.12.1

Explain the difference between the `throws` clause and the `throw` statement.

The **throws** clause states that a method may throw the specified checked exceptions. The **throw** statement is how an exception is thrown.

A.12.3

When would it be better to throw an exception rather than catch it in a method?

If the method has the ability (available information) to handle an exception, then it should catch it and perform the appropriate handling. If the method cannot handle an exception completely, then after catching it, it should throw it or a related exception. In this case, when throwing the exception it should include the original exception as a parameter to the newly thrown exception's constructor.

A.12.5

For the following situations, indicate whether it would be better to catch an exception, declare an exception, or throw an exception in the lower-level method. Explain your answer and show the code required for the lower-level method to do it.

a. A lower-level method contains a call to method `readLine`; the higher-level method that calls it contains a `catch` clause for class `IOException`.

b. A method contains a call to method `readLine` to enter a value that is passed as an argument to a lower-level method. The lower-level method's argument must be a positive number.

c. A lower-level method contains a call to method `readLine`, but the higher-level method that calls it does not have a `catch` clause for class `IOException`.

d. A lower-level method reads a data string and converts it to type `int`. The higher-level method contains a `catch` clause for class `NumberFormatException`.

e. A lower-level method detects an unrecoverable error that is an unchecked exception.

a. The lower-level method should declare IOException. The caller is expecting it.

b. This method should contain a catch clause within a loop for NumberFormatException to ensure that the input value is a valid number. It should also catch IOException. The catch clause should print a message and call System.exit since this is not generally a recoverable error.

c. It should catch IOException. The catch clause should print a message and call System.exit since this is not generally a recoverable error.

d. No special code is required for NumberFormatException, but It should catch IOException. The catch clause should print a message and call System.exit since this is not generally a recoverable error.

e. A lower-level method detects an unrecoverable error that is an unchecked exception.

e. No special code is required.