Chapter 1

1. Yes, since it is not a public class. However, you should name your java file the same as the class.
2. No, a Java source file must have the **java** extension.

Chapter 2

1. ASCII stands for American Standard Code for Information Interchange.
2. Java uses both ASCII characters and Unicode characters. It uses ASCII characters for almost all input elements, except comments, identifiers, and the contents of characters and strings. Java uses Unicode characters for comments, identifiers and the contents of characters and strings.
3. A reference type variable holds the reference (address) of an object, a primitive type variable is used to store a primitive.
4. By using the final keyword.
5. An expression is a legal combination of Java operators and operands that gets evaluated.
6. Examples of Java operators:
   - unary minus operator (-).
   - increment operator (++)
   - decrement operator (--)
   - logical complement operator (!)
   - bitwise complement operator (~)
   - addition operator (+)
   - subtraction operator (-)
   - modulus operator (%)
   - AND conditional operator (&&)
   - OR conditional operator (||)
   - left shift operator (<<)

7. A ternary operator operates on three operands. The ? : operator is the only ternary operator in Java.
8. The operator precedence indicates which operation is performed first in the presence of multiple operators in an expression.
9. Traditional comments and end-of-line comments.

Chapter 3

1. An expression is a legal combination of Java operators and operands that gets evaluated. A statement is an instruction to do something.
2. You can escape from a **while** loop by using the break statement. For example:

```
while (true) {
    // statements
    if (expression) {
        break;
    }
}
```

3. No. The following lines of code have the same effect.

```
for (int x = 0; x < length; x++)
for (int x = 0; x < length; ++x)
```

4. This will be printed.

```
One player is playing this game.
Two players are playing this game.
```

Chapter 4

1. Constructors, methods, fields.
2. A constructor is used to construct an object. A method is used to perform an action. A constructor does not have a return value, and, as you will see in Chapter 6, “Inheritance,” methods are inherited but constructors are not.
3. No.

4. A null reference variable is not referencing any object.
5. To refer to the current object from a method or a constructor.
6. No. Applying the equal operator == to reference variables compare the addresses to objects, not the contents of the objects.
7. Variable scope refers to the accessibility of a variable.
8. Technically, a variable that has been destroyed or no longer accessible.
9. By checking if the object is still being referenced.
10. Having more than one method with the same name in the same class.

Chapter 5

1. The state of an immutable object cannot change. **String** objects are immutable and manipulating a **String** object results in a new **String** instance.
2. Prior to JDK 5, by using **System.in.read()** as in the **getUserInput** method below

```
public String getUserInput() {
    StringBuilder sb = new StringBuilder();
    try {
        char c = (char) System.in.read();
        while (c != '\r') {
            sb.append(c);
            c = (char) System.in.read();
        }
    } catch (IOException e) {
    }
    return sb.toString();
}
```

   In Java 5, by using the **next** method of the **java.util.Scanner** class.
3. Yes, the wrapper classes are the templates for creating wrapper objects, so these wrapper classes are still required. In addition, wrapper classes posses methods that can be used for parsing and formatting.
4. You cannot resize an array, but you can create another array and copy the contents of the first array to the new one.

5. Varargs is a feature in Java 5 that allows methods to have a variable length of argument list.

Chapter 6

1. No.
2. Because of the "is a" relationship between a subclass and a superclass. An instance of a subclass can therefore be assigned to a superclass variable.
3. Method overloading is a feature in many OOP language that allows methods in the same class to have the same name. Method overriding is an OOP feature that enables you to change the behavior of a method in a subclass. In method overloading, the signatures of the methods must not be the same. In method overriding, the signatures of the methods must be identical.
4. So that you can call a method in the parent class and not overridden in the subclass.

Chapter 7

1. The **try** statement provides an easy way of error handling. The alternative to this strategy is a series of **if** statements that tests each of the conditions that might lead to an error. Using the latter is harder and may make your code hard to read.

Chapter 8

1. Simple and complex mathematical operations.
2. **java.util.Date**.
3. **java.text.SimpleDateFormat**.

Chapter 9

1. Because thinking of an interface as a class without implementation misses the big picture. An interface defines methods that both the service provider and its client must agree on.
2. A concrete class that provides default implementations of an interface.
3. An interface that can provide limited implementation.
4. Base classes and abstract classes look similar but their reasons for existence are different, albeit similar.

Chapter 10

1. The enum can be part of a class or it can stand alone. For the latter, you write it as you would a class. For example:

```
public enum CustomerType {
    INDIVIDUAL,
    ORGANIZATION
}
```

2. Enums are safer than static final fields as enumerated values because they can restrict values. On the other hands, with static final fields you use **int**s and can assign any value of **int**.

Chapter 11

1. **Collection**, **List**, **Set**, **Queue**, **ArrayList**, **Vector**, **Comparator**, **Map**, **HashMap**, **Hashtable**.
2. **ArrayList** is unsynchronized, **Vector** is synchronized.
3. **Comparator** is more powerful than **Comparable** because with **Comparator** you can compare objects in more than one way.

Chapter 12

1. Generics impose stricter type checking at compile time and eliminates most type castings.
2. A parameterized type is a generic type.

Chapter 13

1. A stream connects Java code to a data reservoir.
2. **InputStream**, **OutputStream**, **Reader**, **Writer**.
3. Storing objects to persistent storage, such as a file.
4. The class must implement **java.io.Serializable**.

Chapter 14

1. A nested class is a class declared within the body of another class or interface. An inner class is a type of nested class, a non-static one.
2. You can use nested classes to completely hide an implementation. Anonymous classes provide for a shorter way of writing event listeners.
3. A class that does not have a name.

Chapter 15

1. The ability of the JVM to invoke the correct method implementation when a superclass variable is assigned an instance of a subclass.
2. Where the type of object is not known at compile time.

Chapter 16

1. An annotation type is a type of annotation objects. Technically, an annotation type is a special type of interface. Annotations are instances of annotation types.
2. An annotation type for annotation annotations.
3. **Override**, **Deprecated**, and **SuppressWarnings**.

Chapter 17

1. Create different versions of the parts with static contents.
2. By creating a different properties file for each locale.
3. **java.util.Locale** and **java.util.ResourceBundle**.

Chapter 18

1. Because it is very hard to deal with data streams directly at the hardware level.
2. The protocol, the host, the port, and the path.
3. java.net.URL.
4. A socket is an endpoint of a network connection. A socket enables an application to read from and write to the network. Two software applications residing on two different computers can communicate with each other by sending and receiving byte streams over a connection.
5. A server socket is used in a server application and its primary task is to wait for connections. For each connection obtained, a server socket creates a socket to communicate with the remote computer making the connection.

Chapter 19

1. The smallest unit of processing.
2. Protecting a critical section so that only one thread at a time can access an object's critical sections.

3. Code segments that guarantee only one thread at a time have access to a shared resource.