Parameter Passing (primitive types)

- The act of passing an argument takes a copy of a value and stores it in a local variable accessible only to the method which is called.

```java
int num1 = 38;
void myMethod(int num2) {
    num2 = 50;
}
myMethod(num1);
```

- Before:
  - num1 38
  - num2 38
- After:
  - num1 38
  - num2 50

Parameter Passing (objects)

- Objects (in this case arrays) are also passed by value. In this case, however, the value is the address of the object pointed to by the reference variable.

```java
int[] a = {5, 7};
void myMethod(int[] b) {
    b[0] = 5;
    b[1] = 7;
}
myMethod(a);
```

- Before:
  - a 5 7
  - b 0 0
- After:
  - a 10 7
  - b 5 7

Objects and Reference Variables

```
acct1
acctNumber 72354
balance 102.56
name Ted Murphy
```
```
acct2
acctNumber 69713
balance 40.00
name Jane Smith
```

In the previous example there is only one array and two references to it.
The array can be modified through either reference.

```java
int[] array = {10, 7};
```

Method Overloading

- The compiler determines which method is being invoked by analyzing the parameters

```java
float tryMe(int x)
{
    return x + .375;
}

float tryMe(int x, float y)
{
    return x*y;
}
```

Invocation

```java
result = tryMe(25, 4.32)
```

Method Overloading

- The `println` method is overloaded:

```java
println(String s)
println(int i)
println(double d)
```

and so on...

- The following lines invoke different versions of the `println` method:

```java
System.out.println("The total is:");
System.out.println(total);
```

Inheritance

- Inheritance is a fundamental object-oriented design technique used to create and organize reusable classes

  - Here is a quick analogy

    Donald Duck
What can be inherited in Java?

In class hierarchies the Inheritance arrow usually points up instead of down.
Inheritance

- Inheritance allows a software developer to derive a new class from an existing one
- The existing class is called the parent class, or superclass, or base class
- The derived class is called the child class or subclass
- As the name implies, the child inherits characteristics of the parent
- That is, the child class inherits the methods and data defined by the parent class

Class Hierarchy

```
class Car extends Vehicle
{
    // class contents
}
```

Objects

```
Vehicle v1 = new Vehicle();
Car c1 = new Car();
Car c2 = new Car();
Car c3 = new Car();
```

Inheritance relationships are shown in a UML class diagram using a solid arrow with an unfilled triangular arrowhead pointing to the parent class.

- Proper inheritance creates an is-a relationship, meaning the child is a more specific version of the parent.

A programmer can tailor a derived class as needed by adding new variables or methods, or by modifying the inherited ones.

Software reuse is a fundamental benefit of inheritance.

By using existing software components to create new ones, we capitalize on all the effort that went into the design, implementation, and testing of the existing software.

- In Java, we use the reserved word extends to establish an inheritance relationship.

Book & Dictionary Example

- See Words.java (page 440)
- See Book.java (page 441)
- See Dictionary.java (page 442)
The protected Modifier

- Visibility modifiers affect the way that class members can be used in a child class
- Variables and methods declared with private visibility cannot be referenced by name in a child class
- They can be referenced in the child class if they are declared with public visibility -- but public variables violate the principle of encapsulation
- There is a third visibility modifier that helps in inheritance situations: protected
The protected Modifier

- The protected modifier allows a child class to reference a variable or method directly in the child class.
- It provides more encapsulation than public visibility, but is not as tightly encapsulated as private visibility.
- A protected variable is visible to any class in the same package as the parent class.
- The details of all Java modifiers are discussed in Appendix E.
- Protected variables and methods can be shown with a # symbol preceding them in UML diagrams.

The super Reference

- Constructors are not inherited, even though they have public visibility.
- Yet we often want to use the parent’s constructor to set up the “parent’s part” of the object.
- The super reference can be used to refer to the parent class, and often is used to invoke the parent’s constructor.

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Class</th>
<th>Interfaces</th>
<th>Method</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>protected</td>
<td>The class name is the same as the parent class name.</td>
<td>The method is declared in the child class.</td>
<td>N/A</td>
<td>Protected variables and methods can be shown with a # symbol preceding them in UML diagrams.</td>
</tr>
<tr>
<td>final</td>
<td>The class is not extended.</td>
<td>The method is declared in the child class.</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>native</td>
<td>N/A</td>
<td>No method body is required.</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>abstract</td>
<td>N/A</td>
<td>The method must be implemented.</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>static</td>
<td>N/A</td>
<td>The method can be invoked without an instance of the class.</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>synchronized</td>
<td>N/A</td>
<td>The method can be invoked by only one thread.</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>transient</td>
<td>N/A</td>
<td>The variable is not visible outside the class.</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>volatile</td>
<td>N/A</td>
<td>The variable can be accessed by multiple threads.</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Appendix E

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Class and Interfaces</th>
<th>Methods and variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>private</td>
<td>Visible only in this class.</td>
<td>Visible to any class in the same package as the class.</td>
</tr>
<tr>
<td>protected</td>
<td>N/A</td>
<td>Visible to any class in the same package as the class.</td>
</tr>
<tr>
<td>public</td>
<td>Visible anywhere.</td>
<td>Visible anywhere.</td>
</tr>
</tbody>
</table>

The super Reference

- A child’s constructor is responsible for calling the parent’s constructor.
- The first line of a child’s constructor should use the super reference to call the parent’s constructor.
- The super reference can also be used to reference other variables and methods defined in the parent’s class.
Multiple Inheritance

Java supports single inheritance, meaning that a derived class can have only one parent class.

- Multiple inheritance allows a class to be derived from two or more classes, inheriting the members of all parents.
- Collisions, such as the same variable name in two parents, have to be resolved.
- Java does not support multiple inheritance.
- In most cases, the use of interfaces gives us aspects of multiple inheritance without the overhead.

Modified Book Example

- See `Words2.java` (page 445)
- See `Book2.java` (page 446)
- See `Dictionary2.java` (page 447)
This example shows how multiple inheritance can be faked in Java.

```
java.lang.Object
  \<<interface>>
  java.lang.Runnable
    \<<extends>>
    java.lang.Thread
```

THE END