Inheritance -- Introduction

Another fundamental object-oriented technique is called inheritance, which, when used correctly, supports reuse and enhances software designs.

Chapter 8 focuses on:
- the concept of inheritance
- the protected modifier
- adding and modifying methods through inheritance
- creating class hierarchies
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.
- The derived class is called the child class, or subclass.
- The parent class inherits the methods and data defined for the parent class.
- As the name implies, the child inherits characteristics of the parent class.

Inheritance
Inheritance relationships are often shown graphically, with the arrow pointing to the parent class:

- The child is-a more specific version of the parent.

Inheritance should create an is-a relationship, meaning:

- The child is-a specific version of the parent.
Deriving Subclasses

• In Java, the reserved word `extends` is used to establish an inheritance relationship.

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

See Words.java.
The details of each modifier are given in Appendix F.

- The protected visibility modifier allows a member to be inherited, but provides more protection than public does.
- But public variables violate our goal of encapsulation.
- The protected visibility modifier allows a member to be inherited, and those with protected visibility are not.
- Variables and methods declared with protected visibility are not.
- The visibility modifiers determine which class members get inherited and which do not.
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 is often used to invoke the parent’s constructor.

See Words2.java
See Eating.java and School.java.

• Defined vs. Inherited

- Defined for that child
  - A subtle feature of inheritance is the fact that even if a method or variable is not inherited by a child, it is still defined for that child. And can be referenced indirectly through parent methods.
  - But even members that are not inherited exist for the child class, as if they were declared in the child class.

- An inherited member can be referenced directly in the child class, and can be referenced indirectly through parent methods.

• A subtle feature of inheritance is the fact that even if a method or variable is not inherited by a child, it is still defined for that child.

• Defined vs. Inherited
Overriding Methods

A child class can override the definition of an inherited method in favor of its own.

That is, a child can redefine a method it inherits from its parent's method, but can have different code in the body.

The new method must have the same signature as the parent's method, but can have different code in the body.

The object type determines which method is invoked.

See Messages.java.
Overloading vs. Overriding

• Don’t confuse the concepts of overloading and overriding

• Overloading deals with multiple methods in the same class with the same name but different signatures

• Overriding deals with two methods, one in a parent class and one in a child class, that have the same signature

• Overloading lets you define a similar operation in different ways for different data

• Overriding lets you define a similar operation in different ways for different object types

• Overloading lets you define a similar operation in different ways for different data types
The super reference can be used to invoke any method from the parent class. This ability is often helpful when using overridden methods.

The syntax is:

`super.method(parameters)`

See `Firm.java` and `Accounts.java`.
A child class of one parent can be the parent of another:

Class Hierarchies
Class Hierarchies

- Two children of the same parent are called **siblings**.
- Good class design puts all common features as high in the hierarchy as is appropriate for all situations.
- There is no single class hierarchy that is appropriate for to keep up with changing needs.
- Class hierarchies often have to be extended and modified.
- The hierarchy as is reasonable.
- Good class design puts all common features as high in
- Two children of the same parent are called **siblings**.

See Accounts2.java.
The Object Class

- All objects are derived from the Object class.
- If a class is not explicitly defined to be a child of an existing class, it is assumed to be the child of the Object class. Therefore, the ultimate root of all class hierarchies is the Object class.
- The Object class contains a few useful methods, such as `toString()`, which are inherited by all classes.

See `TestToString.java`.
References and Inheritance

- An object reference can refer to an object of its class, or
- To an object of any class related to it by inheritance

For example, if the `Holiday` class is used to derive a child class called `Christmas`, then a `Holiday` reference could actually be used to point to a `Christmas` object:

```java
Holiday day;
day = new Christmas();
```
References and Inheritance

• Assigning a predecessor object to an ancestor reference is considered to be a widening conversion, and can be performed by simple assignment.

• Assigning an ancestor object to a predecessor reference can also be done, but it is considered to be a narrowing conversion and must be done with a cast.

• The widening conversion is the most useful.
Polymorphism

- A polymorphic reference is one which can refer to one of several possible methods.

Polymorphism

If `day` refers to a `Holiday` object, it invokes `Holiday`'s version of `celebrate`; if it refers to a `Christmas` object, it invokes `Christmas`'s version of `celebrate`.

Now consider the following invocation:

```java
day.celebrate();
```

Suppose the `Holiday` class has a method called `celebrate`, and the `Christmas` class overrode it.

• A polymorphic reference is one which can refer to one of several possible methods.
Polymorphism

Polymorphic references are therefore resolved at run-time, not during compilation.

Each polymorphic reference could execute different methods at different times. If an invocation is in a loop, the exact same method is invoked on each iteration.

Note that, if the object referenced by the reference is an instance of a subclass, the method is invoked which is appropriate for that object.

See Messages2.java.
Polymorphism

• Note that, because all classes inherit from the Object class, an Object reference can refer to any type of object.

• A Vector is designed to store Object references.

• The instanceof operator can be used to determine the class from which an object was created.

• See Variety.java.
Polymorphism

- See Firm2.java.