Introduction to Inheritance

CS 3: Computer Programming in Java
Objectives

- Introduce inheritance, including
  - What it is
  - Its properties
  - Controlling it
- Talk about the protected modifier
- Discuss the super reference
- Look at references and inheritance
- Delve into the Object class and the this reference
- Take a peek into polymorphism
What Is Inheritance?

- Another fundamental object-oriented technique is called inheritance, which enhances software design and promotes reuse of code.

- Inheritance focuses on:
  - Deriving new classes
  - Creating class hierarchies
  - The protected modifier
  - Polymorphism via inheritance
Properties of Inheritance

- Inheritance allows a software developer to derive a new class from an existing one.
- The existing class is called the parent class.
  - Sometimes known as the base class or superclass (preferred in Java).
- The derived class is called the child class.
  - Sometimes known as the subclass (in Java).
Properties of Inheritance (2)

- As the name implies, the child class inherits characteristics of the parent class
  - That is, the child class inherits the methods and data defined for the parent class
- Inheritance should create an is-a relationship, meaning the child is a more specific version of the parent
Controlling Inheritance

- Visibility modifiers determine which inherited class members are visible within the child class and which are not.
- Parent class variables and methods declared with public visibility are inherited visible and those with private visibility are not.
- But public variables violate our goal of encapsulation!
- So, there is a third visibility modifier that helps in inheritance situations: protected.
The protected Modifier

- The protected visibility modifier allows a member of a base class to be inherited into the child.
- It allows access from any method of any class in the same package (as though it was declared public).
- The protected visibility provides more encapsulation than does public.
- However, protected visibility is not as tightly encapsulated as private visibility.
  - Private variables are inherited, but can only be accessed thru the protected or public access methods of the superclass.
The super Reference

- Unfortunately, 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.
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.

- Assigning a subclass object to a superclass reference is considered to be a widening conversion and can be performed by simple assignment.

  - In other words, making a parent class reference point to a child class object is ok because remember that the child class is a base class object and it just has the stuff written up for its class and what is written for the base class.
Assigning a superclass object to a subclass reference can also be done, but it is considered to be a narrowing conversion and must be done with a cast.

In other words, making a child class reference point to a parent class object is not generally ok because the child class encompasses more stuff than is defined in the base class.

But the compiler will let you do it if you cast the type to match the child class.

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.
The Object class

- A class called Object is defined in the java.lang package of the Java standard class library
- It is at the top of the Class Hierarchy
- All classes are derived from the Object class
- If a class is not explicitly defined to be the child of an existing class, it is assumed to be the child of the Object class
- The Object class is therefore the ultimate root/parent of all class hierarchies
The Object class contains a few useful methods, which are inherited by all classes. For example, the `toString` method is defined in the Object class. Every time we have defined `toString`, we have actually been overriding it! The `toString` method in the Object class is defined to return a string that contains the name of the object’s class and a hash value.
The Object class (3)

- That’s why the `System.out.println` method can call `toString` for any object that is passed to it - all objects are guaranteed to have a `toString` method via inheritance.

- A child class can override the definition of an inherited method in favor of its own definition.
  - That is, a child can redefine a method that it inherits from its parent.
  - The new method must have the same signature as the parent's method, but can have different code in the body.
  - The type of the object executing the method determines which version of the method is invoked.
The equals method of the Object class determines if two references are aliases

You may choose to override equals to define equality in some other way for your class objects

Many classes provide ways to compare objects of their class to each other by overriding the equals method

- The String class, for instance, has an equals method so that the contents of two String objects can be compared
The Object class and the this Reference Example

If we wanted to add an equals method to the Employee class that would compare the employeeName of two Employee objects, it would look like the code below:

```java
public boolean equals(Object e) {
    if (this.employeeName.equals(((Employee)e).employeeName))
        return true;
    else
        return false;
}
```
The this Reference

- Note the use of the `this` reference in the if statement so that the name in the object that called the method can be differentiated from the name in the passed object.
- The calling object is using the `equals` method from the `String` class to compare the two names.
- The `this` reference is a valuable tool when working inside a method that handles two objects from the same class.
Introduction to Polymorphism

- When the type of object assigned to a reference is determined at runtime, and that, in turn, determines which version of a method is invoked, it is called polymorphism.
- If a method is passed an object of the base class type as its argument, then any reference to any child class object can also be passed thru that argument.
- Why? This is because a child “is a” object of the same type as its parent (and more).
- So, Java determines at runtime what type of object is actually being referenced, and then calls the method from the class of reference.
Introduction to Polymorphism (2)

- In this program, we introduce a Manager class which also inherits (extends) from the Employee class.
- It differs from the Technician class in the way the getHolidays() method is written.
- A Manager gets an extra holiday for every year of service above 10 years, whereas a Technician gets an extra holiday for every year of service above 5 years.
Now, when a reference to a Technician is passed to argument Employee person in the helper method:

```java
static void displayDetails(Employee person)
{
    System.out.println(person+"n");
    System.out.println("Holidays:"+person.getHolidays()+"n\n");
}
```

Java will determine at runtime that the reference is really a Technician reference, and call the getHolidays() method from the Technician class.

Likewise, if a Manager reference is passed to argument Employee person in the helper method, Java will determine at runtime that the reference is really a Manager reference, and call the getHolidays() method from the Manager class.
Summary

- Inheritance focuses on
  - Deriving new classes
  - Creating class hierarchies
  - The protected modifier
  - Polymorphism via inheritance

- Properties of inheritance
  - The existing class is called the parent class
  - The derived class is called the child class
  - The child class inherits the methods and data defined for the parent class
Summary (2)

- Can control inheritance through the **protected** modifier
  - Allows a member of a base class to be inherited into the child
  - Allows access from any method of any class in the same package (as though it was declared public)
  - Provides more encapsulation than does public
  - However, not as tightly encapsulated as private visibility

- Discussed the **super** reference
  - Can be used to refer to the parent class, and is often used to invoke the parent's constructor
Summary (3)

- Looked at 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

- Delved into the Object class and the this reference
  - The Object class is therefore the ultimate root/parent of all class hierarchies
  - The this reference is a valuable tool when working inside a method that handles two objects from the same class

- Took a peek into polymorphism
  - When the type of object assigned to a reference is determined at runtime, and that, in turn, determines which version of a method is invoked