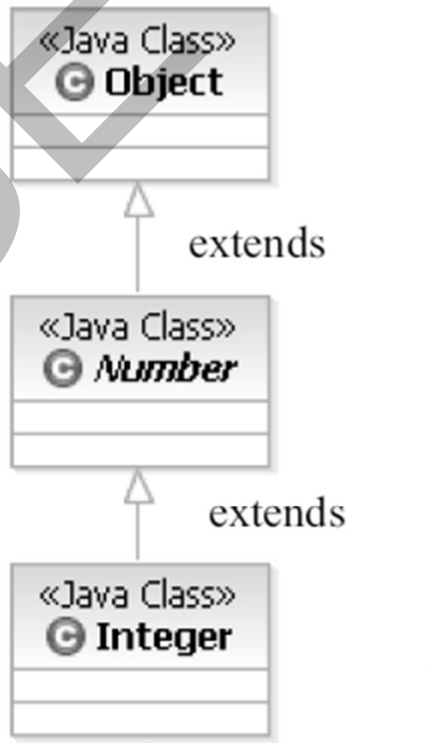
Unit objectives

- After completing this unit, you should be able to:
  - Describe the inheritance of fields and methods
  - Explain the concept of a class hierarchy
  - Outline how subclasses specialize superclasses
  - Explain how method lookup works
  - Create and use subclasses
  - Describe how polymorphism is implemented

Class hierarchies

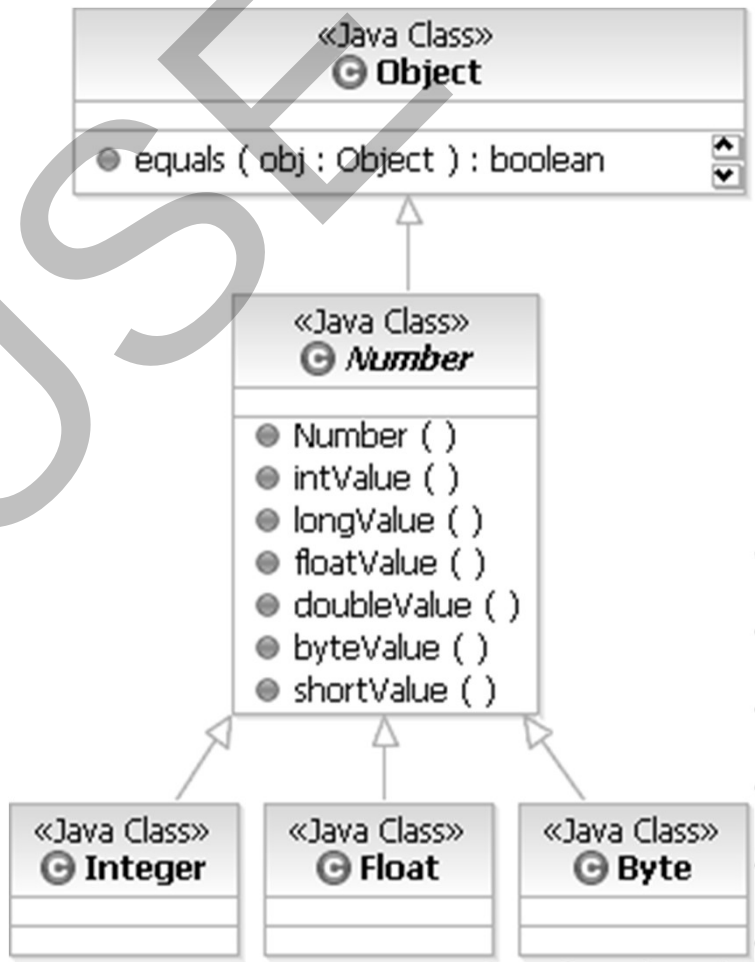
- Every object belongs to a class
  - The objects that are members of a class are its instances
- Every class (except Object) has a superclass
  - In Java, Object is the root of the entire class hierarchy
- When defining new classes, the developer must decide which class is the appropriate superclass




```
Integer zero = new Integer(0);
```

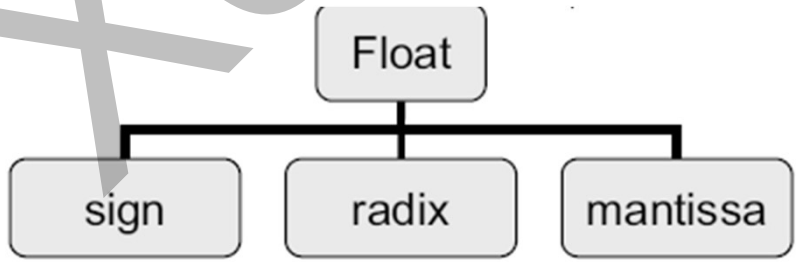
Specialization and generalization

- A subclass is a *specialization* of its superclass
  - Specialize the state and behavior in the subclass by adding fields and extending or changing methods
- A superclass is a *generalization* of its subclasses
  - Common state and behavior can be moved to the superclass where it is available to all subclasses
  - Code is written once, and maintained in one place



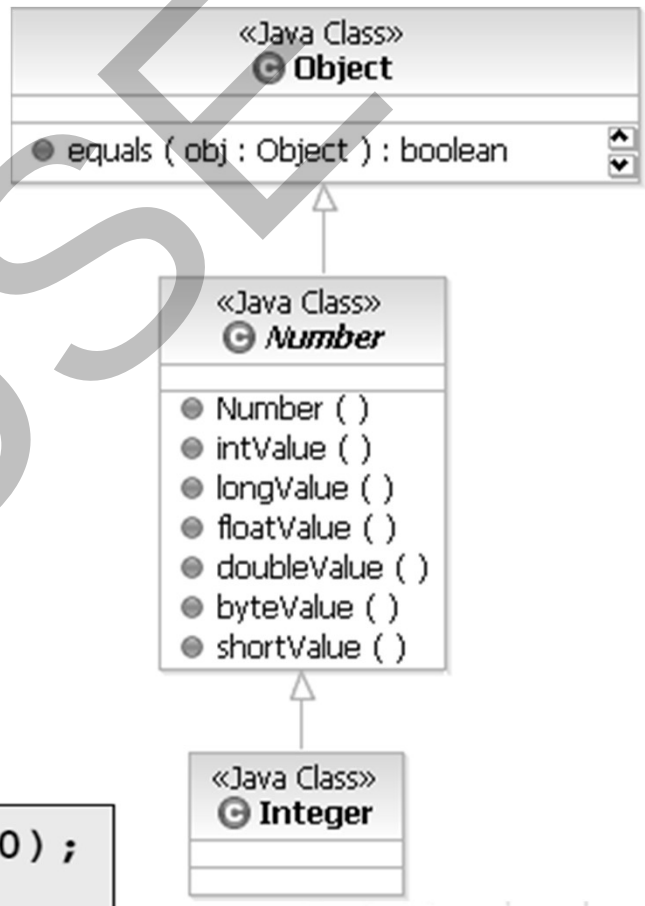
Inheritance relationships

- Inheritance describes relationships where an instance of the subclass is a special case of the superclass
  - Inheritance relationships are called "***is a***" relationships because the subclass "is a" special case (specialization) of the superclass
  - Examples: A floating point number *is a* number; a number *is an* object
- Inheritance cannot describe "*has a*" relationships
  - This is usually done using the class' fields and their associated methods
    - Example: A floating point number *has a* sign, a radix and a mantissa
  - Collections provide generalized ways to handle “has a” relationships



Inheriting fields and methods

- Each subclass inherits the fields of its superclass
  - These fields in the superclass may have been inherited from classes even further up in the class hierarchy
- Each subclass inherits the methods of its superclass
  - An object will understand all messages which its class has implemented or its superclass has either inherited or implemented




```
Integer zero = new Integer(0);
if (zero.equals(x)) {
   byte b = zero.byteValue();
   …
}
```

Access modifiers

- Variables and methods in Java have access restrictions, described by the following access modifiers:
  - **private**
    - Access is limited to the class in which the member is declared
    - Example: private int x;
  - **default** (this means no modifier is used)
    - Access is limited to the package in which the member is declared
    - Example: int x;
  - **protected**
    - Access is limited to the package in which the member is declared, as well as all subclasses of its class
    - Example: protected void setName() { . . . }
  - **public**
    - The member is accessible to all classes in all packages
    - Example: public String getName() { . . . }

Overriding methods

- You can extend or change superclass behavior by overriding the inherited method in the subclass
- To override a superclass's method, create a new method in the subclass with the same signature (name and parameter list)
  - Java uses the new method in place of the inherited one
  - This new method replaces or refines the method of the same name in the superclass
  - If the signatures are different, the Java interpreter will consider the superclass and the subclass method to be distinct

```
public class MyClass extends Object {
   public boolean equals(Object o) {
      if (o==null)
      …
   }
}
```

Restrictions on overriding methods

- Restrictions on the new method:
  - The parameter list must match the inherited method exactly
  - The return type must be the same as that of the inherited method
  - The access modifier must not be more restrictive than that of the inherited method
    - For example, if overriding a protected method, the new method can be protected or public, but not private

Example of overriding

```
public class BankAccount {
    private float balance;
    public float getBalance() {
       return balance;
    }
}

public class InvestmentAccount
                    extends BankAccount {
    private float cashAmount;
    private float investmentAmount;
    public float getBalance() {
       return cashAmount + investmentAmount;
    }
}
```

Method lookup

- The compiler looks up the implementation for a method call beginning in the object's class definition
- If the method is not found in the object's own class, the search continues in the superclass and up the hierarchy until it is found
- When the method is found, it is invoked on the object to which the message was passed
- If the method was never implemented in any of the classes in the hierarchy, an error is issued at compile time

Inheritance and static methods

- A class can call all static methods defined in its superclass as though they were defined in the class itself
- Static methods can be hidden by static methods in the subclass

```
superclass
static String t = "test";
public static String superTest(String s) {
   s += " was the arg.";
   return s;
}
```

```
subclass
public static void main(String[] args){
   System.out.println(superTest(t));
}
```

Inheritance and constructors

- Only constructors within the class being instantiated and within the immediate superclass can be invoked
- A constructor can call another constructor in its superclass using the keyword super and the parameter list
  - The parameter list must match that of an existing constructor in the superclass
- Constructors in the same class are invoked with the keyword this and the parameter list
- The first line of your constructor can be one of:
  - super(…);
  - this(…);

The superclass in object construction

- Superclass objects are built before the subclass
  - The compiler supplies an implicit super() call for all constructors
  - super(…) initializes superclass members
- If the first line of your constructor is not a call to another constructor, super() is called automatically
  - Zero-argument constructor in the superclass is called as a result
  - This can cause an error if the superclass does not have a zero-argument constructor

Default constructors

- If you do not provide any constructors, a default zero-argument constructor is provided for you
  - The default zero-argument constructor just makes a call to super()
- If you implement any constructor, Java will no longer provide you with the default zero-argument constructor
  - You can write your own zero-argument constructor which behaves like the default constructor (that is, just makes an implicit call to super())

More on this and super

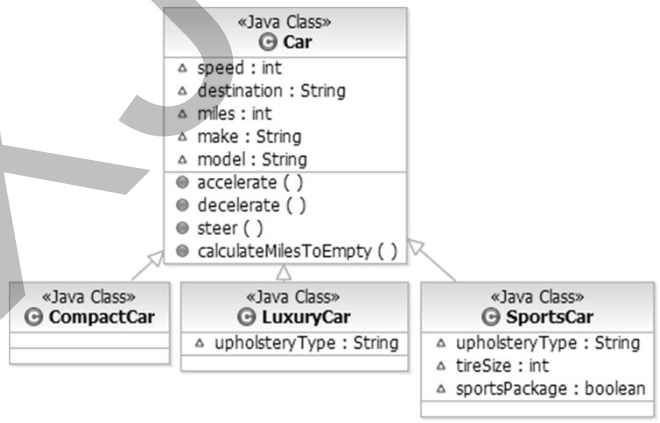
- this and super can be used in instance methods and constructors in Java
  - They cannot be used in class (static) methods
- this and super affect method lookup
  - this: method lookup starts in the current class
  - super: method lookup starts in the immediate superclass
- The keyword super lets you use code from a superclass when overriding a method in the current class
  - When a subclass overrides a superclass method, calls to that method will go to the code that overrides the superclass method
  - Using super allows you to look up methods starting from the immediate superclass

Pollymorphiism

- A compact car, luxury car, and sports car are several classes of car that accept the same set of messages, and provide the same service
  - Each car accepts the accelerate(), decelerate(),steer() and calculateMilesToEmpty() messages,allowing you to drive to a destination
- The service may be implemented differently by each car, but these classes may be interchanged without affecting the driver who sends messages to the vehicle
  - This principle is known as *polymorphism*

Polymorphism through inheritance

- In Java, classes that inherit from the same superclass inherit the same methods, and can respond to the same messages
  - CompactCar, LuxuryCar and SportsCar can respond to the same messages, and be interchanged without affecting the message sender, a driver
  - Inheritance is one way that Java implements polymorphism
  - See the inheritance relationship between classes that model these cars:



Implementing polymorphism

- A variable can be assigned objects of its declared type, or subtypes of its declared type
  - For example:
    - Car auto = new Car();
    - Car auto = new CompactCar();
    - Car auto = new LuxuryCar();
    - Car auto = new SportsCar();
- Assigning an object of one type to an object of another type (higher in the hierarchy), will make the object forget its real type
  - From above, **auto** will no longer know that it is an object of class **CompactCar,** and will only respond to messages for the **Car** class
  - You can get these objects to remember their real type by casting the object to that type, such as:
    - CompactCar cc = (CompactCar)auto;
  - Casting to an unrecognized subclass will throw a **ClassCastException**

Example

- Since the variables forget their type once declared with the type of the superclass, they will only respond to messages in the Car type
- They can respond to the same messages and may be interchanged without affecting the message sender

```
public class Driver {
  public static void main(String[] args) {
     Car auto = new CompactCar();
     auto.accelerate();
     if(stopLight.equals("red") {
          auto.decelerate();
          auto.accelerate();
     }
     if(corner == true) {
          auto.steer();
     }
  }
}
```

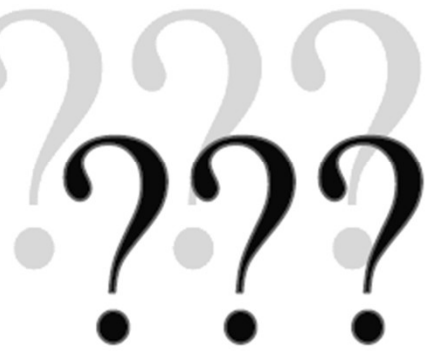
Exercise

A company pays its employees on a weekly basis. The employees are of four types: Salaried employees are paid a fixed weekly salary regardless of the number of hours worked, hourly employees are paid by the hour and receive overtime pay (i.e., 1.5 times their hourly salary rate) for all hours worked in excess of 40 hours, commission employees are paid a percentage of their sales and base-salaried commission employees receive a base salary plus a percentage of their sales. For the current pay period, the company has decided to reward salaried-commission employees by adding 10% to their base salaries. The company wants to write an application that performs its payroll calculations polymorphically.

Checkpoint

- 1. What are “is-a” and “has-a” relationships, and what do they have to do with class hierarchies?
- 2. What are the four Java access modifiers, and how do they limit the scope of class members?
- 3. What is wrong with the following code?

```
public class MyClass extends MySuper {
    MyClass() {
      super();
    }
    MyClass(int i) {
      super();
      this();
    }
}
```



Unit Summary

- In this unit, you should have learned to:
  - Describe the inheritance of fields and methods
  - Explain the concept of a class hierarchy
  - Outline how subclasses specialize superclasses
  - Explain how method lookup works
  - Create and use subclasses
  - Describe how polymorphism is implemented