Inheritance

A **base** class (**superclass** or **parent** class) defines some generic behavior.

A derived class (**subclass** or **child** class) can **extend** the base class.

A subclass inherits all of the functionality of the base class and may also define additional functionality specific to the child class.

A subclass has an "**is-a**" relationship with the parent class.

A parent class can have many subclasses, but a subclass can only extend **one** parent class. However, a subclass can implement many interfaces.

The data members in the parent class are usually defined with the **protected** visibility modifier, instead of private, to ensure that the data is visible to its subclasses. The protected modifier exposes the data members to other classes in the package.

***Note**: If the keyword extends is not used to define a class, that class implicitly extends the Object class. Class **A** below **extends the Object class**.*

```
public class A {
     protected int x;
}

public class B extends A {
     private int y;
}
```

Abstract Class

An abstract class is a class that has 1 or more abstract methods.

```
public abstract class A {
     protected int x, y;

     public abstract int methodA();

     public int larger(A other){
          if(methodA() > other.methodA())
               return x;
          return y;
     }
}
```

Interface

An interface is similar to a fully abstract class. It consists of public abstract methods and public static final fields, only.

A class that *implements* an interface must provide definitions for all of the abstract methods in the interface. Otherwise, the class must be declared as abstract.

The visibility of the inherited method from the interface must be public.

While a class can extend only one other class, it may implement many

interfaces. Just like a class, the **IS-A** relationship holds.

The ***instanceof*** operator can be used to determine if a reference is type-compatible with an interface.

When a class implements an interface method, it must implement the exact signature; otherwise, it inherits the abstract version from the interface, and the method written is an overloaded version of the method.

A class may not implement two interfaces that contain a method with the same signature and incompatible return types, since it would be impossible to provide both methods in one class.

If a class fails to implement all of the methods inherited from an interface, it must be declared abstract.

Interfaces can extend other interfaces (including multiple interfaces).

```
public interface Comparable {
      int compareTo(Object o);
}

public class BlackJackCard implements Comparable {
      private int rank, suit;
      public int compareTo(Object o) {
            int val = -1;
            if(o instanceof BlackJackCard)
                  val = rank - ((BlackJackCard)(o)).rank;
            return val;
      }
}
```

Polymorphism

Literally means “many forms”. An object variable can take on different forms.

A superclass reference variable can refer to any of its descendants.

Dynamic/Late Binding:

At run-time, Java determines which version of an inherited method is called by evaluating the contents of the object, not the reference.

```
Staff staff = new Staff("Mary Smith", "N00975310","C3078",49,35);

Employee emp = staff;
System.out.println(emp.pay());
```

The following is an invalid statement because Employee is an abstract class. **Employee**

```
employee = new Employee("Mary Smith", "N00975310", "C3078");
```

Casting:

If a method exists in the subclass, but not the superclass, a reference of the superclass type cannot call that method. Even though *p* refers to a Staff type, *p.pay()* is undefined.

```
Person p = staff;
System.out.println(((Staff)(p)).pay());
```

```java
public class InheritanceApp {

   public static void main(String[] args) {
      Faculty faculty = new Faculty("Dave Watson", "N00543210","B3038",2000);
      Staff staff = new Staff("Mary Smith", "N00975310","C3078",49,35);
      Student student = new Student("John Doe", "N00123456", 2.54);

      System.out.println(faculty);
      System.out.println(student);
      System.out.println(staff);

      // A polymorphic reference
      Person[] person = new Person[] {faculty, staff, student};

      // Late or dynamic binding
      for(int i=0; i < person.length; i++)
           System.out.println(person[i]);
   }
}
```

Output from the program above:

```
*****************************
Name: Dave Watson
ID: N00543210
Email Address: Dave.Watson@qc.cuny.edu
Weekly Salary: 2000.0
*****************************
Name: John Doe
ID: N00123456
Email Address: John.Doe@qc.cuny.edu
*****************************
Name: Mary Smith
ID: N00975310
Email Address: Mary.Smith@qc.cuny.edu
Weekly Salary: 1715.0
*****************************
Name: Dave Watson
ID: N00543210
Email Address: Dave.Watson@qc.cuny.edu
Weekly Salary: 2000.0
*****************************
Name: Mary Smith
ID: N00975310
Email Address: Mary.Smith@qc.cuny.edu
Weekly Salary: 1715.0
*****************************
Name: John Doe
ID: N00123456
Email Address: John.Doe@qc.cuny.edu
```

```
class Person {
   protected String name, id;

   public Person(String n, String i){
      name = n;
      id = i;
   }
   public String toString(){
      String str = "*****************************";
      str += "\nName: " + name + "\nID: " + id;
      str += "\nEmail Address: " + email();
      return str;
   }
   public String email(){
      return name + "@qc.cuny.edu";
   }
}
/*****************************************************************************/
abstract class Employee extends Person {
   protected String office;

   public Employee(String n, String i, String
      o){ super(n,i);
      office = o;
   }
   public abstract double pay();

   public String toString(){
      String str = super.toString();
      str += "\nWeekly Salary: " +
      pay(); return str;
   }
}
/*****************************************************************************/
class Student extends Person {
   protected double gpa;
   public Student(String n, String i, double
      g){ super(n, i);
      gpa = g;
   }
}
```

```
class Faculty extends Employee {
  private double salary = 1500;

  public Faculty(String n, String i, String o, double
    s){ super(n, i, o);
    salary = s;
  }
  public double pay(){
    return salary;
  }
}
/*********************************************************************************/
 class Staff extends Employee {
  private double rate, hours;

  public Staff(String n, String i, String o, double r, double h){
    super(n, i, o);
    rate = r;
    hours = h;
  }
  public double pay() {
    return hours * rate;
  }
}
```