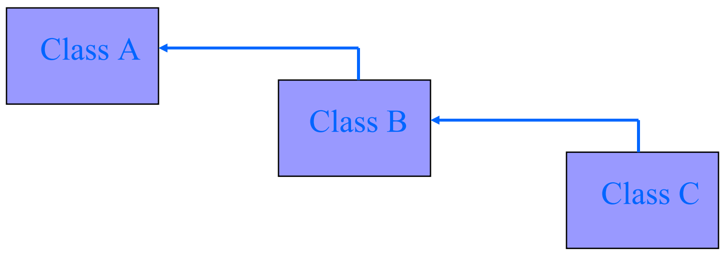
# Inheritance

## Inheritance

- Defining a new class based on an existing class is called derivation
- The derived class is also called the direct subclass of the base or super class
- You can also derive classes from the derived class and so on



## Inheritance

- class B extends A{

  //definition of class B

  }
- **The keyword extends identifies that class B is a direct subclass of class A**
- **The class B can have additional members in addition to the inherited members of class A**
- **What are the inherited members?**
  - **The member which is accessible in the derived class**

## Example – Super Class

```
public class Animal
{
  public Animal(String aType)
  {
    type = new String(aType);
  }

  public String toString()
  {
    return "This is a " + type;
  }

  private String type;
}
```

## Example – Sub Class

```
public class Dog extends Animal
{
  public Dog(String aName)
  {
    super("Dog");                    // Call the base constructor
    name = aName;                            // Supplied name
    breed = "Unknown";                       // Default breed value
  }

  public Dog(String aName, String aBreed)
  {
    super("Dog");                            // Call the base
constructor
    name = aName;                            // Supplied name
    breed = aBreed;                          // Supplied breed
  }

  private String name;                       // Name of a Dog
  private String breed;                      // Dog breed
}
```

## Example TestDerived

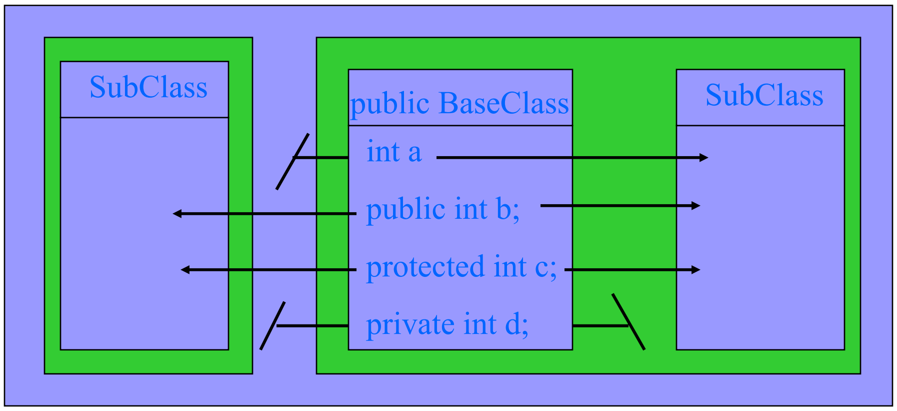
```
public class TestDerived
{
  public static void main(String[] args)
  {
    Dog aDog = new Dog("Fido", "Chihuahua");     // Create a dog
    Dog starDog = new Dog("Lassie");     // Create a Hollywood dog
    System.out.println(aDog);            // Let's hear about it
    System.out.println(starDog);                 // and the star
  }
}
```

## Inheritance

- The inclusion of members of a base class in a derived class so that they are accessible in that derived class is called class inheritance
- An inherited member of a derived class is a full member of that class and is freely accessible to any method in the class
- Which members of the base class are inherited?

## Inheriting Data Members

- The inheritance rules apply to class variables as well as instance variables



## Inheritance

- You can define a data member in a derived class with the same name as data member in the base class.
- The data member of the base class is still inherited but is hidden by the derived class member with the same name
- The hiding will occur irrespective if the type or access modifiers are the same or not.
- Any use of the derived member name will always refer to the member defined in derived class
- To refer to the inherited base class member, you must qualify it with the keyword super

## Inheriting Methods

- Methods in a base class excluding constructors are inherited in a derived class in the same way as the data members of the base class
- Methods declared as private in a base class are not inherited
- Default access methods are only inherited if you define the derived class in the same package as the base class
- Note: Constructors in the base class are never inherited regardless of their attributes
- Though the base class constructors are not inherited in your derived class, you can still call them or if you don't call a base class constructor from your derived class constructor, the compiler will try to do it for you
- The super class constructor is called in a subclass using super ( );

## Inheriting Methods

- The super/base class constructor call must be the first statement in the body of the derived class constructor
- If the first statement in a derived class constructor is not a call to a base class constructor, the compiler will insert a call to the default class constructor i.e super ( ), for you
- Default call for base class constructor is with no arguments. This sometimes result in a compiler error.Why?
  - When you define your own constructor in a class, no default constructor is created by the compiler. Thus you have to define the no argument constructor for the class yourself so that in a derived class you don't get the compile error due to call to default constructor of base class

## Overriding Methods

- You can define a method in a derived class that has the same signature as a method in the base class. In this case the method defined in the derived class overrides the method in the base class
- The base class method is still present in the derived class and it is possible to call the base class method also
- An overridden method can be called from within the subclass using super.xx( ) where xx( ) is the method name

## Example2

```
public class Dog extends Animal
{
  public Dog(String aName)
  {
    super("Dog");                          // Call the base constructor
    name = aName;                          // Supplied name
    breed = "Unknown";                     // Default breed value
  }
  public Dog(String aName, String aBreed)
  {
    super("Dog");                          // Call the base constructor
    name = aName;                          // Supplied name
    breed = aBreed;                        // Supplied breed
  }
  // Present a dog's details as a string
  public String toString()
  {
    return "It's " + name + " the " + breed;
  }

  private String name;                     // Name of a Dog
  private String breed;                    // Dog breed
}
```

## Overriding Methods

- **Same signature and return type**
- **Each parent class method may be overridden at most once in any one subclass**
- **You cannot change the access attributes**
  - **method cannot be made more restrictive than that of the base class method it overrides**

## Overriding vs Overloading

- **Overloaded methods supplement each other while an overriding method replaces the method it overrides**
- **Overloaded methods can exist in any number in the same class while each method in a base class can be overridden at the most once in any one class**
- **Overloaded methods must have different parameter lists while overriding methods must have identical type and order of the parameter list**
- **The return type of an overloaded method can be chosen freely while the return type of an overriding method must be identical to that of the method it overrides**

## Polymorphism

- **Poly means many and morph many forms. The word polymorphism means the ability to assume different forms or shapes.**
- **In programming terms, it is the ability of a single reference variable of a given type to be used to reference objects of different types and to automatically call the method that is specific to the type of object the variable references**
- **So a single method call behaves differently depending on the type of the object which the call references**

## Polymorphism

- **We can store a reference to an object in a variable of the same type**
- **We can store a reference to a derived class object in a variable of the derived class type AND *we can also store it in a variable of any direct or indirect base class***
- **To be able to support polymorphic behavior, there are a few requirements**
  - The method call for the derived class object must be through a variable of a base class object
  - The method called must also be a member of the base class
  - The method being used must satisfy all requirements for overridden methods

## Example – Polymorphism

```
public class Animal
{
  public Animal(String aType)
  {
    type = new String(aType);
  }

  public String toString()
  {
    return "This is a " + type;
  }

// Dummy method to be implemented in the  derived classes
  public void sound()
  {}

  private String type;
}
```

## Example – Polymorphism

```
public class Cat extends Animal
{
  public Cat(String aName)
  {
    super("Cat");          // Call the base constructor
    name = aName;          // Supplied name
    breed = "Unknown";     // Default breed value
  }

  public Cat(String aName, String aBreed)
  {
    super("Cat");          // Call the base constructor
    name = aName;          // Supplied name
    breed = aBreed;        // Supplied breed
  }
```

## Example – Polymorphism

```
// Return a String full of a cat's details
  public String toString()
  {
    return super.toString() + "\nIt's " + name + " the " + breed;
  }

  // A miaowing method
  public void sound()
  {
    System.out.println("Miiaooww");
  }

  private String name;      // Name of a cat
  private String breed;     // Cat breed
}
```

## Example – Polymorphism

```
public class Dog extends Animal
{
  public Dog(String aName)
  {
    super("Dog");                            // Call the base constructor
    name = aName;                            // Supplied name
    breed = "Unknown";                       // Default breed value
  }
  public Dog(String aName, String aBreed)
  {
    super("Dog");                            // Call the base constructor
    name = aName;                            // Supplied name
    breed = aBreed;                          // Supplied breed
  }
  // Present a dog's details as a string
  public String toString()
  {
    return super.toString() + "\nIt's " + name + " the " + breed;
  }

  // A barking method
  public void sound()
  {
    System.out.println("Woof    Woof");
  }
  private String name;                       // Name of a Dog
  private String breed;                      // Dog breed
}
```

## Example – Polymorphism

```
public class Duck extends Animal
{
  public Duck(String aName)
  {
    super("Duck");         // Call the base constructor
    name = aName;          // Supplied name
    breed = "Unknown";     // Default breed value
  }
  public Duck(String aName, String aBreed)
  {
    super("Duck");         // Call the base constructor
    name = aName;          // Supplied name
    breed = aBreed;        // Supplied breed
  }
  // Return a String full of a duck's details
  public String toString()
  {
    return super.toString() + "\nIt's " + name + " the " + breed;
  }
  // A quacking method
  public void sound()
  {
    System.out.println("Quack quackquack");
  }
  private String name;      // Duck name
  private String breed;     // Duck breed
}
```

## Example – Polymorphism

```
import java.util.Random;
public class TryPolymorphism
{
  public static void main(String[] args)
  {
    // Create an array of three different animals
    Animal[] theAnimals = {
                            new Dog("Rover", "Poodle"),
                            new Cat("Max", "Abyssinian"),
                            new Duck("Daffy","Aylesbury")
                          };

    Animal petChoice;                   // Choice of pet

    Random select = new Random();       // Random number generator
    // Make five random choices of pet
    for(int i = 0; i < 5; i++)
    { // Choose a random animal as a pet
      petChoice = theAnimals[select.nextInt(theAnimals.length)];
      System.out.println("\nYour choice:\n" + petChoice);
      petChoice.sound();                        // Get the pet's reacti
    }
  }
}
```

## Polymorphism

- When you call a method using a variable of a base class type, polymorphism results in the method that is called being selected based on the type of the object stored, not the type of the variable
- A variable of base type can store a reference to an object of any derived type, the kind of object stored is not known until the program executes. Thus the choice of which method to execute has to be made dynamically when the program is running
- Your program can adapt at runtime to accommodate and process different kinds of data quite automatically

## Polymorphism

- Polymorphism only applies to methods
- When you access a data member of a class object, the variable type always determines the class to which the data member belongs

## Universal Superclass

- All classes you define are subclasses by default
- All classes have a standard class Object as the base class
- You don't have to specify this, its implicit
- As Object is a super class of all classes, variable of type Object can hold reference to an object of any class and is useful to handle objects of unknown types
- Object class don't have any data members, only member methods

## Universal Superclass

- toString ( ) // name of class @ hexadecimal code
- equals( )  // compares objects
- getClass( )   // final and returns object of type Class
- hashCode( ) // returns hash code value for object in int
- notify( )   (final)
- notifyAll( )   (final)
- wait( )   (final)
- clone( )
- finalize( )