**Inheritance**

Extending Classes

* Inheritance
  + Superclass – Parent class, Base Class
    - The class is giving its members to its child class
  + Subclass – Child class, Derived Class
    - The class taking members from its base class
* Java Platform Class Hierarchy
  + **Object** is at the root of Java Class Hierarchy
* Java supports inheritance through **extends** keyword

class Person { … }

class Student **extends** Person { … }

class Employee **extends** Person { … }

* + Inheritance – Derived Class
    - Class takes all members from another class
      * Class Person ( Mother: Person, Father: Person ) ( Super class)
      * Student ( Mother, Father, School School)
      * Employee(Mother, Father, Organization)
* Using inherited Members – You can access inherited members as usual

class Person { public void **sleep**() { … } }

class Student **extends** Person { … }

class Employee **extends** Person { … }

Student student = new Student();

student.**sleep**();

Employee employee = new Employee();

employee.**sleep**();

* Reusing Constructors
  + Constructors are not inherited
  + Constructors can be reused by the child classes

class Student **extends** Person {

private School school;

public Student(String name, School school) {

**super**(name); **Constructor call should be first**

this.school = school;

}

}

* Thinking about inheritance – Extends
  + Derived class instance contains instance of its base class
* Inheritance has a **transitive relation**

class Person { … }

class Student **extends** Person { … }

class CollegeStudent **extends** Student { … }

**Person (superclass) <- Student <-College Student**

* Multiple Inheritance
  + **In Java there is no multiple inheritance**
  + **Only multiple interfaces can be implemented**
* Access to Base Class Members
  + User the **super** keyword

class Person { … }

class Employee **extends** Person {

void fire(String reasons) {

System.out.println(  
 **super**.name +   
 " got fired because " + reasons);

}

}

* Inheritance and Access modifiers
  + Derived classes can access all public and protected members
  + Derived classes can access default members if in same package
  + Private fields are not inherited in subclasses ( can`t be accessed)

class Person {

**private** String id; -> Can be accessed through other methods

**String** name;

**protected** String address;

**public** void sleep();

}

* Shadowing variables
  + Derived classes can hide superclass variables

class Person { protected **int** weight; }

class Patient extends Person {

protected **float** weight; -> hides int weight

public void method() {

**double** weight = 0.5d;

} hides both

}

* Shadowing Variables – Access
  + Use **super** and **this** to specify member access

class Person { protected int weight; }

class Patient extends Person {

protected float weight;

public void method() {

double weight = 0.5d; **Local variable**

this.weight = 0.6f; **Instance** **member**

super.weight = 1; **base** **class** **member**

}

}

* Overriding Derived Methods
  + A child class can redefine existing methods

public class Person {   
 public void **sleep**() Method in base class must not be final  
 { sout("Person sleeping"); }

}

public class Student extends Person {

**@Override public void sleep() Signature and return type should match** { sout("Student sleeping"); }

}

* Final methods
  + **Final** – defines a method that can`t be overridden
* Final Classes
  + Inheriting from a final classes is forbidden

public final class Animal {

…

}

public class Dog extends Animal { } // Error…

public class MyString extends String { } // Error…

public class MyMath extends Math { } // Error…

* Inheritance Benefits :
  + Abstraction
    - One approach for providing abstraction ( focus on common properties )

Person person = new Person();

Student student = new Student();

List<Person> people = new ArrayList();

people.**add**(person); Person ( Base Class)

people.**add(**student); Student (Derived Class)

**Polymorphism**

* + Extension
    - We can extend a class that we can`t otherwise change

**CustomArrayList** **extends** **ArrayList**

* + - **Duplicate code is error prone**
    - **Reuse classes through extension**
    - **Sometimes the only way**
* Composition
  + Using classes to define classes

class **Laptop** {

**Monitor** monitor;

**Touchpad** touchpad;

**Keyboard** keyboard;

…

}

* Delegation

class Laptop {

Monitor monitor;

void incrBrightness() {

monitor.brighten();

}

void decrBrightness() {

monitor.dim();

}

}

Laptop. incrBrightness();

Laptop. decrBrightness();

* When to use inheritance
  + Classes share IS-A relationship ( Too simplistic )
  + Derived class IS-A-SUBSTITUTE for the base class
  + Share the same role
  + Derived class is the same as the base class but adds a little bit more functionality