Introduction

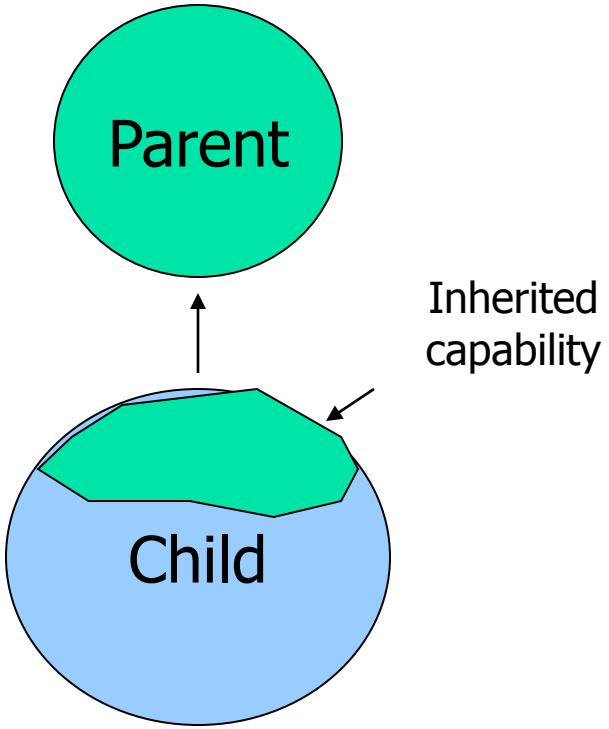
Classes and Subclasses
Or Extending a Class

Inheritance: Introduction

- Reusability--building new components by utilising existing components- is yet another important aspect of OO paradigm.
- It is always good/"productive" if we are able to reuse something that is already exists rather than creating the same all over again.
- This is achieve by creating new classes, reusing the properties of existing classes.

Inheritance: Introduction

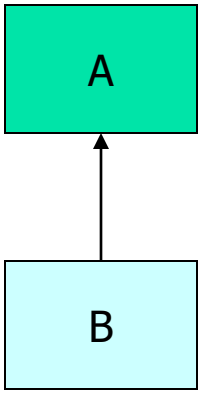
- This mechanism of deriving a new class from existing/old class is called "inheritance".
- The old class is known as "base" class, "super" class or "parent" class"; and the new class is known as "sub" class, "derived" class, or "child" class.



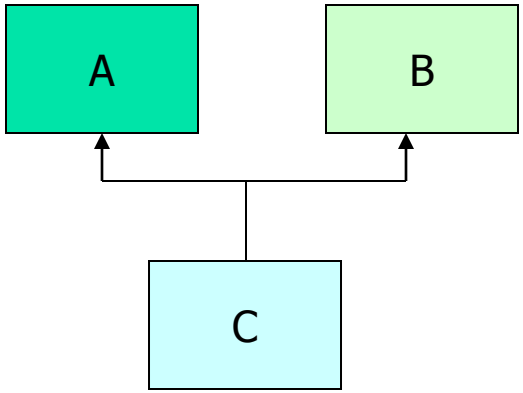
Inheritance: Introduction

- The inheritance allows subclasses to inherit all properties (variables and methods) of their parent classes. The different forms of inheritance are:
  - Single inheritance (only one super class)
  - Multiple inheritance (several super classes)
  - Hierarchical inheritance (one super class, many sub classes)
  - Multi-Level inheritance (derived from a derived class)
  - Hybrid inheritance (more than two types)
  - Multi-path inheritance (inheritance of some properties from two sources).

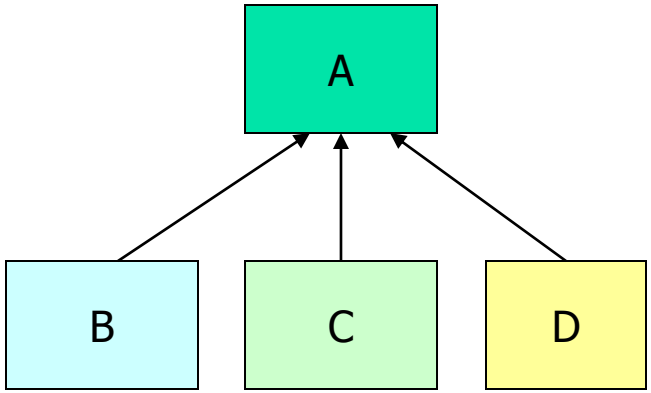
Forms of Inheritance



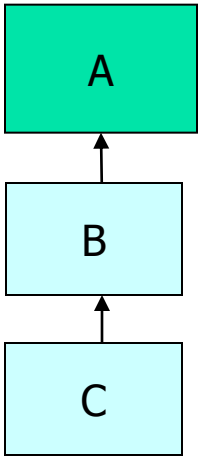
(a) Single Inheritance



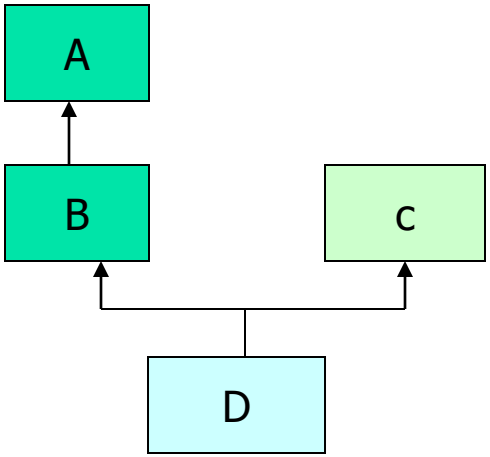
(b) Multiple Inheritance



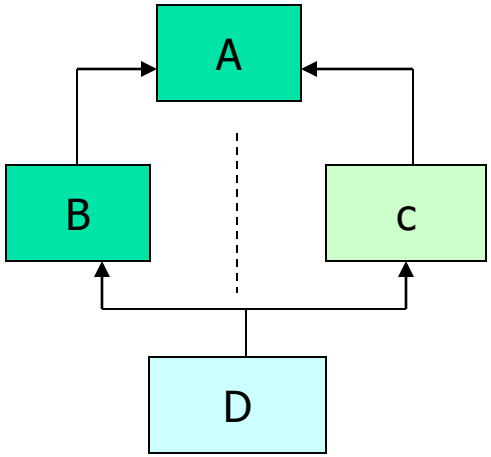
(c) Hierarchical Inheritance



(a) Multi-Level Inheritance



(b) Hybrid Inheritance



(b) Multipath Inheritance

Defining a Sub class

- A subclass/child class is defined as follows:

```
class  SubClassName extends SuperClassName
{
        fields declaration;
        methods declaration;
}
```

- The keyword "extends" signifies that the properties of super class are extended to the subclass. That means, subclass contains its own members as well of those of the super class. This kind of situation occurs when we want to enhance properties of existing class without actually modifying it.

Subclasses and Inheritance

- Circle class captures basic properties
- For drawing application, need a circle to draw itself on the screen, *GraphicCircle*...
- This can be realised either by updating the circle class itself (which is not a good Software Engineering method) or creating a new class that builds on the existing class and add additional properties.

Without Inheritance

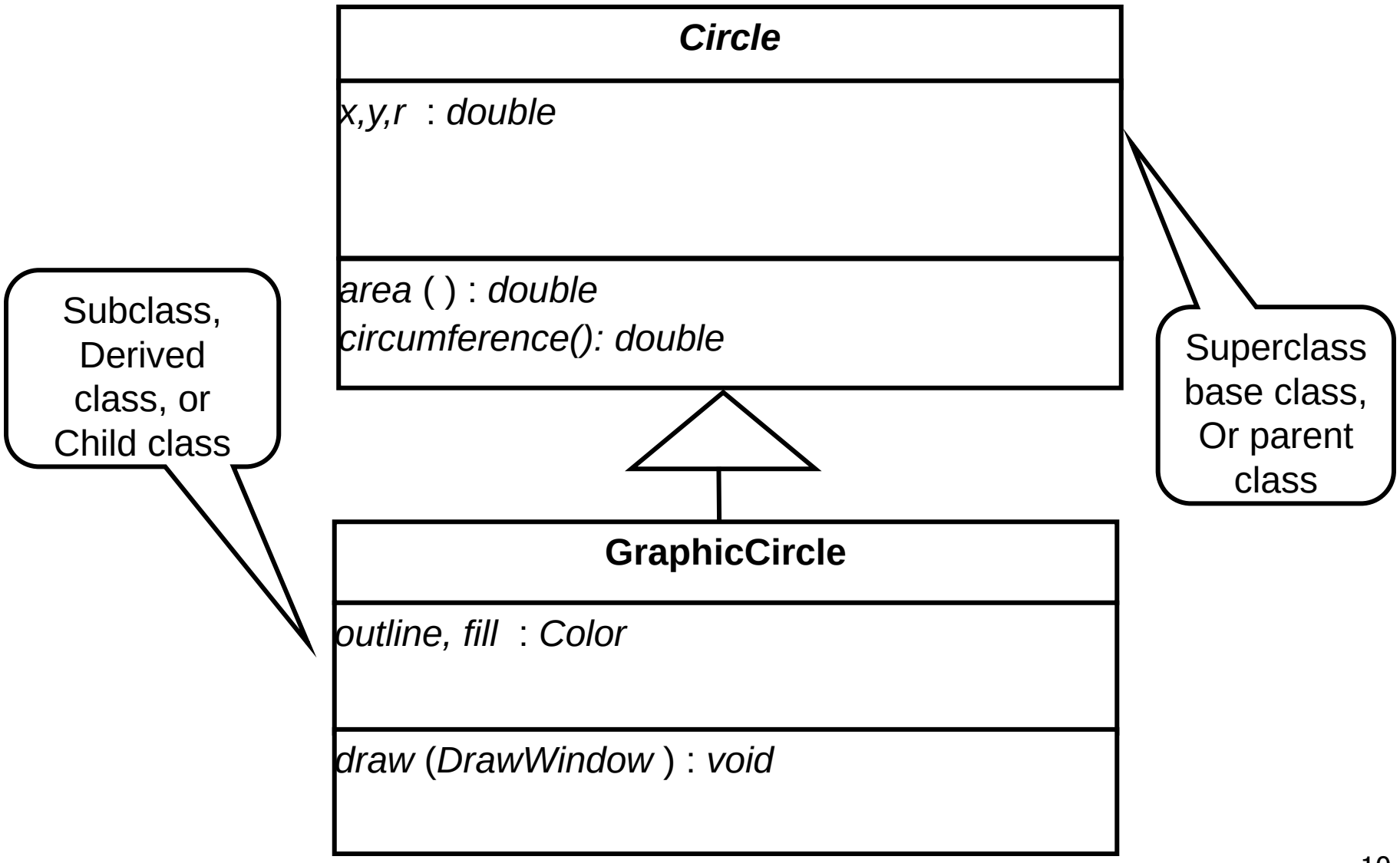
```
public class GraphicCircle {
      public Circle c;  // keep a copy of a circle

      public double area() { return c.area(); }
      public double circumference (){ return c.circumference(); }


     // new instance variables, methods for this class
      public Color outline, fill;
      public void draw(DrawWindow dw) { /* drawing code here */ }
}
```

- Not very elegant

Subclasses and Inheritance

- Circle class captures basic properties
- For drawing application need a circle to draw itself on the screen, *GraphicCircle*
- Java/OOP allows for Circle class code to be implicitly (re)used in defining a *GraphicCircle*
- *GraphicCircle* becomes a *subclass* of Circle, *extending* its capabilities

Subclassing Circle



Subclassing

- Subclasses created by the keyword *extends*:

```
public class GraphicCircle extends Circle {
      // automatically inherit all the variables and methods
      // of Circle, so only need to put in the 'new stuff'

      Color outline, fill;
      public void draw(DrawWindow dw) {
            dw.drawCircle(x,y,r,outline,fill);
      }
}
```

- Each *GraphicCircle* object is also a *Circle*!

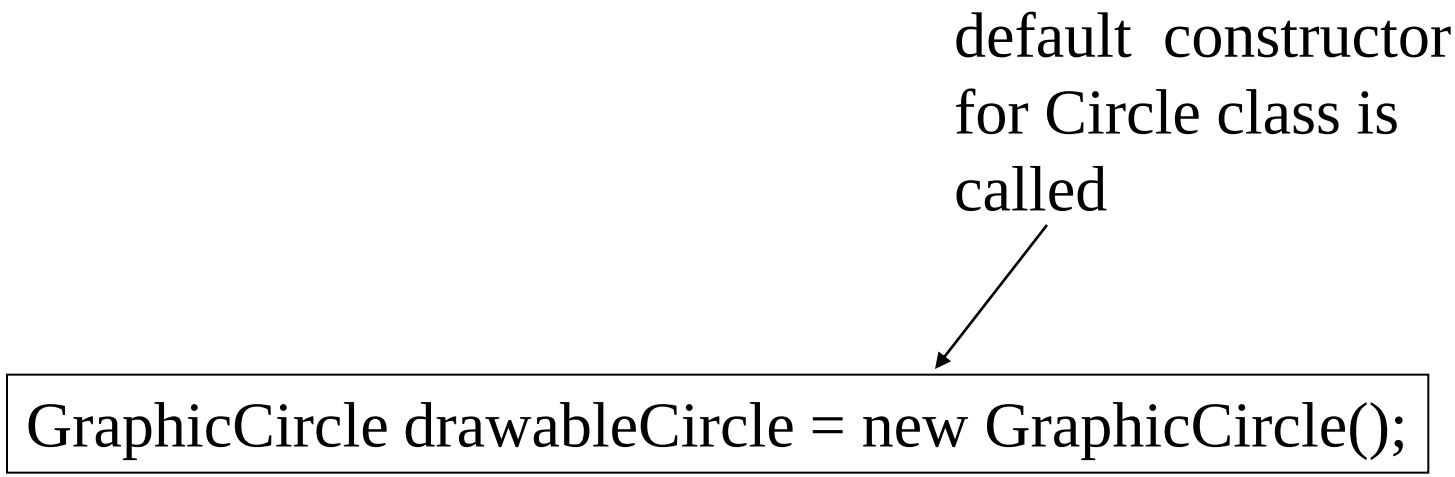
Final Classes

- Declaring class with *final* modifier prevents it being extended or subclassed.
- Allows compiler to optimize the invoking of methods of the class

```
final class Cirlce{
    ..............



}
```

Subclasses & Constructors

- Default constructor automatically calls constructor of the base class:



Subclasses & Constructors

- Defined constructor can invoke base class constructor with *super*:

```
public GraphicCircle(double x, double y, double r,
Color outline, Color fill) {
    super(x, y, r);
    this.outline = outline;
    this fill = fill
}
```

Shadowed Variables

- Subclasses defining variables with the same name as those in the superclass, *shadow* them:

Shadowed Variables - Example

```
public class Circle {
      public  float r = 100;
}

public class GraphicCircle extends Circle {
      public float r = 10;      // New variable, resolution in dots per inch
}

public class CircleTest {
      public static void main(String[] args){
            GraphicCircle gc = new GraphicCircle();
            Circle c = gc;
            System.out.println(" GraphicCircleRadius= " + gc.r);   // 10
            System.out.println (" Circle Radius = " + c.r);              // 100
      }
}
```

Overriding Methods

- Derived/sub classes defining methods with same name, return type and arguments as those in the parent/super class, *override* their parents methods:

Overriding Methods

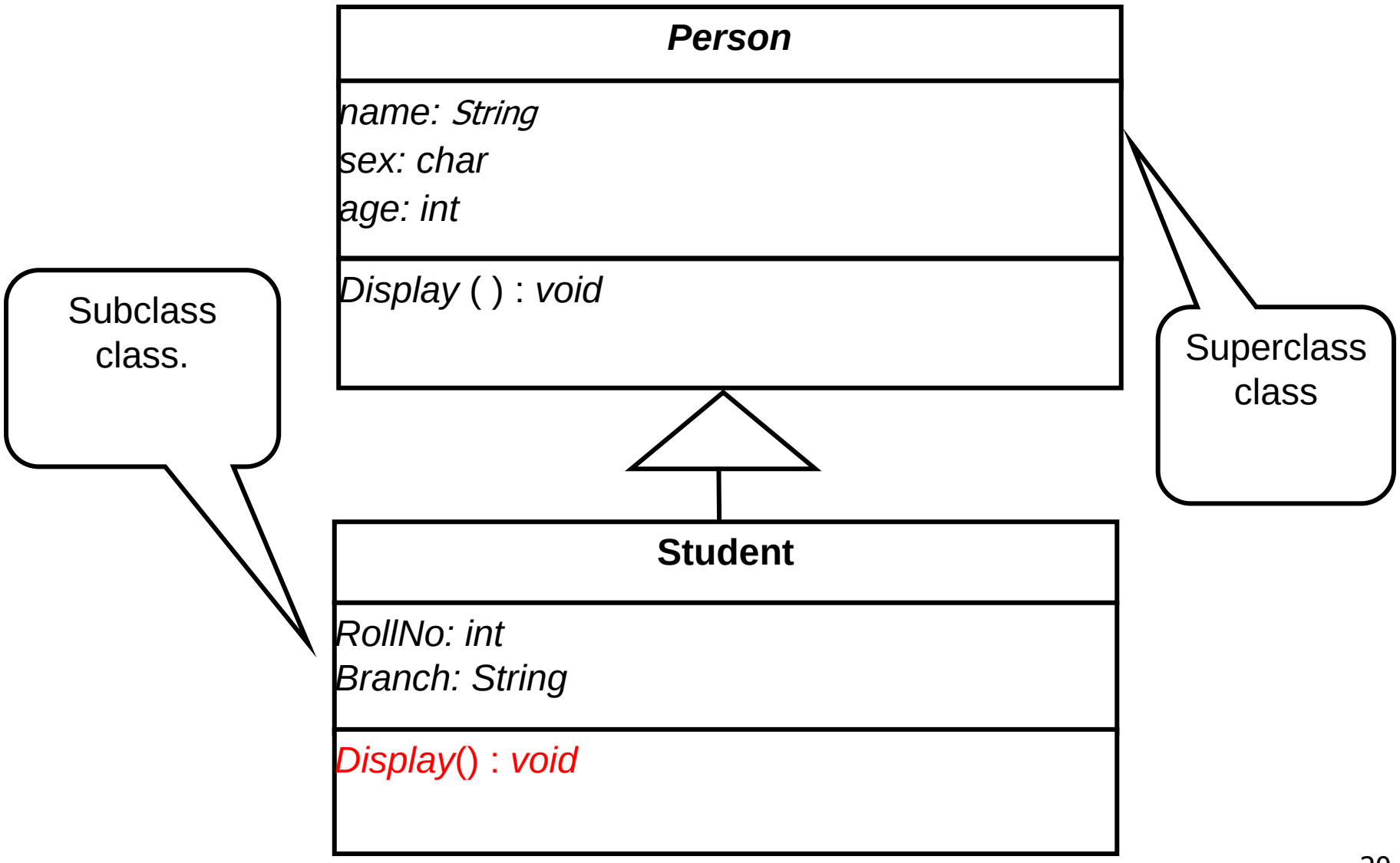
```
class A {
        int j = 1;
        int f( ) { return j; }
}



class B extends A {
        int j = 2;
        int f( ) {
        return  j; }
}
```

Overriding Methods

```
class override_test {
        public static void main(String args[]) {
                B b = new B();
                System.out.println(b.j);        // refers to B.j prints 2
                System.out.println(b.f());      // refers to B.f prints 2
                                                          Object Type Casting
                A a = (A) b;
                System.out.println(a.j);      // now refers to a.j prints 1
                System.out.println(a.f());    // overridden method still refers to B.f() prints 2 !
        }
 }
```

```
[raj@mundroo] inheritance [1:167] java override_test
2
2
1
2
```

Using All in One: Person and Student

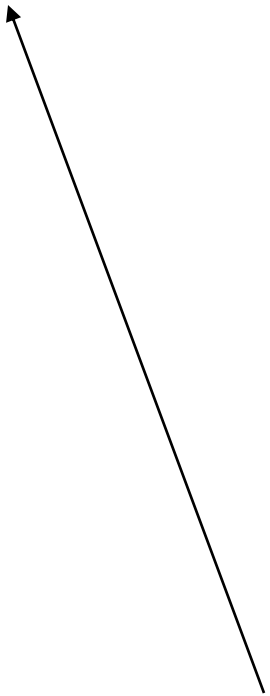


Person class: Parent class

```
// Student.java: Student inheriting properties of person class
class person
{
	private String name;
	protected char sex; // note protected
	public int age;
	person()
	{
		name = null;
		sex = 'U'; // unknown
		age = 0;
	}
	person(String name, char sex, int age)
	{
		this.name = name;
		this.sex = sex;
		this.age = age;
	}
	String getName()
	{
		return name;
	}
	void Display()
	{
		System.out.println("Name = "+name);
		System.out.println("Sex = "+sex);
		System.out.println("Age = "+age);

	}
}
```

Student class: Derived class

```
class student extends person
{
	private int RollNo;
	String branch;
	student(String name, char sex, int age, int RollNo, String branch)
	{
		super(name, sex, age); // calls parent class's constructor with 3 arguments
		this.RollNo = RollNo;
		this.branch = branch;
	}
	void Display()  // Method Overriding
	{
		System.out.println("Roll No = "+RollNo);
		System.out.println("Name = "+getName());
		System.out.println("Sex = "+sex);
		System.out.println("Age = "+age);
		System.out.println("Branch = "+branch);
	}
	void TestMethod() // test what is valid to access
	{
	//	name = "Mark"; Error: name is private
		sex = 'M';
		RollNo = 20;
	}
}
```



What happens if super class constructor is not explicitly invoked ?
(default constructor will be invoked).

Driver Class

```
class MyTest
{
	public static void main(String args[] )
	{
		student s1 = new student("Rama", 'M', 21, 1, "Computer Science");
		student s2 = new student("Sita", 'F', 19, 2, "Software Engineering");

		System.out.println("Student 1 Details...");
		s1.Display();
		System.out.println("Student 2 Details...");
		s2.Display();

		person p1 = new person("Rao", 'M', 45);
		System.out.println("Person Details...");
		p1.Display();

	}
}
```

Can we create Object of person class ?

Output

```
[raj@mundroo] inheritance [1:154] java MyTest
Student 1 Details...
Roll No = 1
Name = Rama
Sex = M
Age = 21
Branch = Computer Science
Student 2 Details...
Roll No = 2
Name = Sita
Sex = F
Age = 19
Branch = Software Engineering
Person Details...
Name = Rao
Sex = M
Age = 45
[raj@mundroo] inheritance [1:155]
```

Summary

- Inheritance promotes reusability by supporting the creation of new classes from existing classes.
- Various forms of inheritance can be realised in Java.
- Child class constructor can be directed to invoke selected constructor from parent using super keyword.
- Variables and Methods from parent classes can be overridden by redefining them in derived classes.
- New Keywords: extends, super, final

Assignment

- Explain different type of inheritance.
- What is derived class.