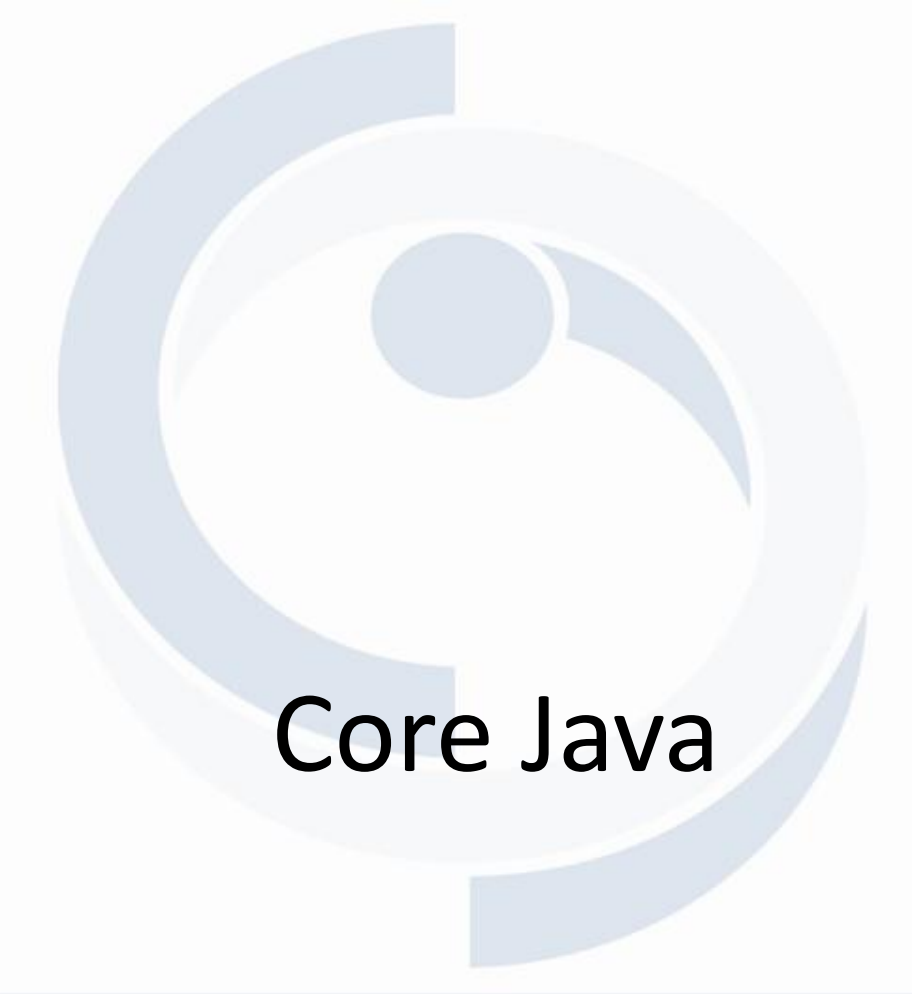

Core Java

Course Objective

At the end of this session, you will be able to:

- **Day 3**
- **Static Members**
- **Inheritance**
- **Runtime Polymorphism**
- **Day 3**
- **Abstraction**
- **Interface**
- **String, StringBuffer, StringBuilder. Date**

Using *static* (Contd…)

- ***static variable***
  - Belongs to a class
  - A single copy to be shared by all instances of the class
  - Creation of instance not necessary for using static variables
- ***static method***
  - It is a class method
  - Accessed using *class name.method name*
  - Creation of instance not necessary for using static methods
  - A static method can access only other static data & methods, and not non-static members

Using *static* (Contd…)

```
Class Student {

  private int rollNo;

  private static int studCount;

  public Student(){

        studCount++;

  }

  public void setRollNo (int r){

       rollNo = r;

  }

  public int getRollNo (int r){

       return rollNo;

  }

 }
```

The static studCount variable is initialized to 0, ONLY when the class is first loaded, NOT each time a new instance is made

Each time the constructor is invoked, i.e. an object gets created, the static variable studCount will be incremented thus keeping a count of the total no of Student objects created

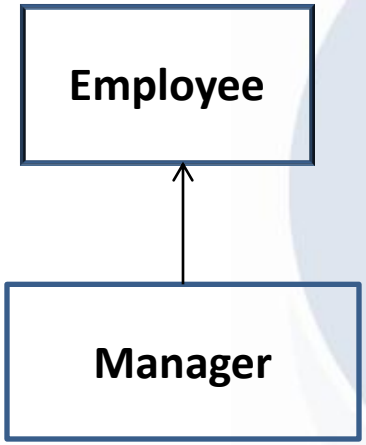
Inheritance

- **Implements IS-A relationship**
- **Capability of a class to use the properties & methods of another class while adding its own functionality**
- **A class derived from another class is called as subclass / derived class / extended class / child class**
- **The class from which the subclass is derived is called as superclass / base class / parent class**
- **Each class is allowed to have one direct superclass**

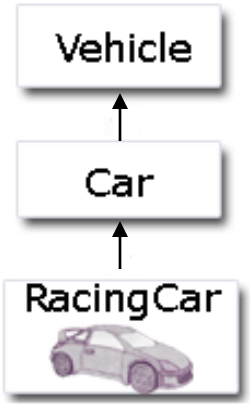
Types of Inheritance

- **Java supports two types of inheritance**

Single Inheritance

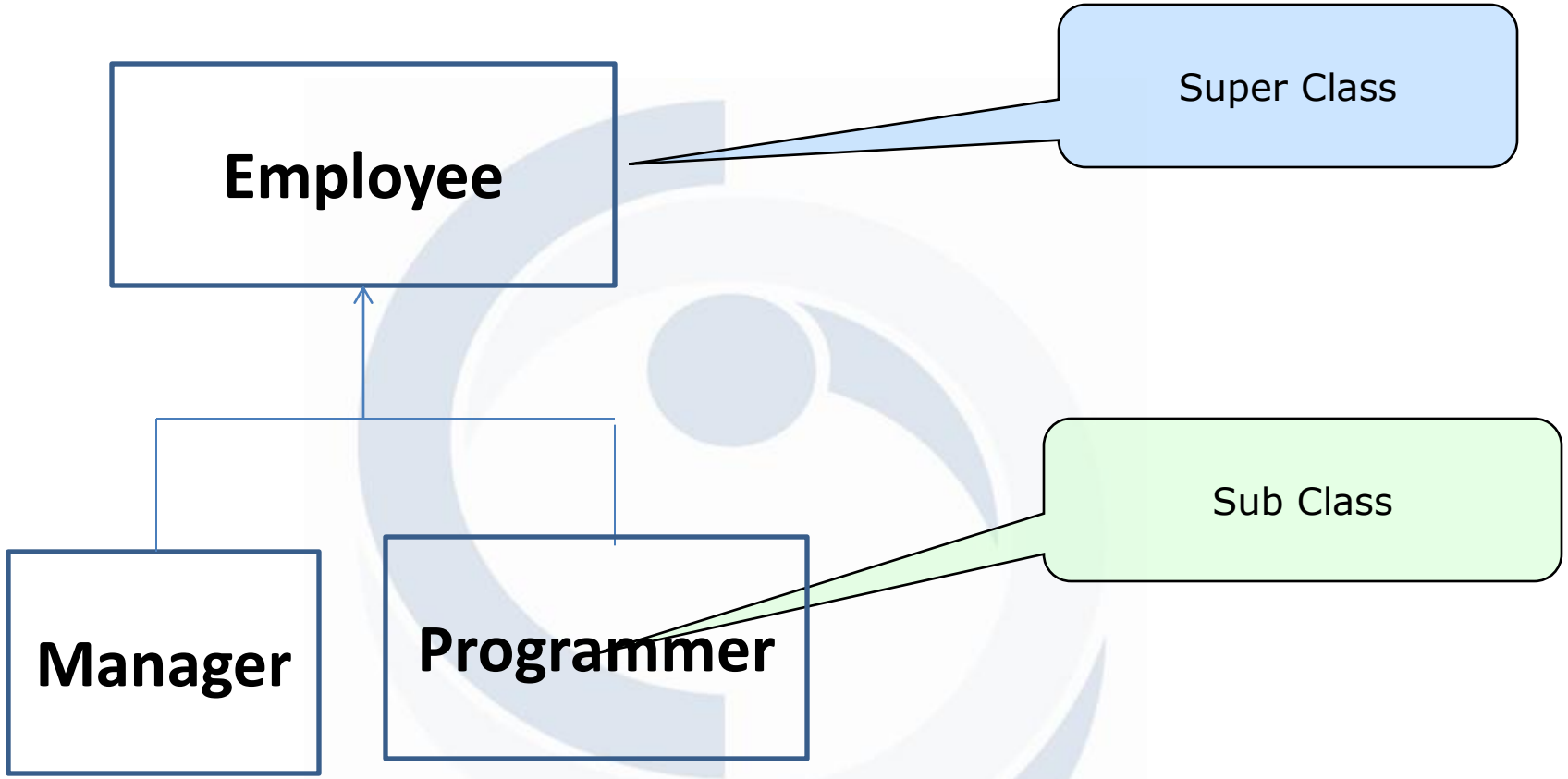


Multilevel Inheritance



Note: Java doesn't support Multiple Inheritance

Inheritance (Contd…)



Inheritance (Contd…)

A class is declared subclass of another class by using the extends keyword

```
class SubClass extends BaseClass{
…}
```

All base class fields & methods are inherited by the subclass

- **private members of the super class are not accessible directly by any method of the subclass**
- **static data members of the base class are also inherited**
- **protected members of the super class are inherited by subclass**

Inheritance (Contd…)

```
public class Employee {
      private String empno;
      private String name;
      private String basicSal;
      private String dept;
}

Class Manager extends Emmployee{
      int no_of_reporting;
      int perks;

}
```

Constructors in Inheritance

- **Constructors are invoked in the order of hierarchy**
- **While instantiating a sub class, its super class default constructor will be invoked first, followed by the sub class constructor**
- **The keyword super can be used to invoke the super class parameterized constructor instead of the default**
- **Remember that:**
  - **super()** call must occur as the first statement in constructor
  - **super()** call can only be used in a constructor definition

Inheritance (Contd…)

```
public class Employee {
      private String empno;
      private String name;
      private int basicSal;
      private String dept;
      public Employee(){}
      public Employee(String empno,String
name,int basicSal,String dept){
     ….  }//parameterized contractor
}
```

Inheritance (Contd…)

```
public class Manager extends Emmployee{
    private int no_of_reporting;
    private int perks;
    public Manager(){}
    public Manager(String empno,String
name,int basicSal,String dept,int
no_of_reporting,int perks){
     super(empno,name,basicSal,dept);
      this.no_of_reporting= no_of_reporting;
      this.perks=perks;
}
}
```

Method Overriding (Contd…)

- **Useful if a derived class needs to have a different implementation of a certain method from that of the superclass**
- **A subclass can override a method defined in its superclass by providing a new implementation for that method**
- **The new method definition must have the same method signature (i.e., method name & parameters) and return type**
- **The new method definition cannot narrow the accessibility of the method, but it can widen it**

Method Overriding

```
public class Employee {
      private String empno;
      private String name;
      private int basicSal;
      private String dept;
      public Employee(){}
public Employee(String empno,String name,int
                     basicSal,String dept){
     ….
 }//parameterized contractor
public void display(){}

}
```

Method Overriding

```
public class Manager extends Emmployee{
    private int no_of_reporting;
    private int perks;
    public Manager(){}
    public Manager(String empno,String name,int
                   basicSal,String dept,int
                   no_of_reporting,int perks){
     super(empno,name,basicSal,dept);
      this.no_of_reporting= no_of_reporting;
      this.perks=perks;
}

   public void display(){   }

}
```

Dynamic Binding

- When an object's class cannot be determined at compile time
- JVM (not the compiler) has to bind a method call to its implementation
- Instances of a sub-class can be treated as instances of the parent class
- So the compiler doesn't know its type, just knows its base type

```
class EmployeeApp{
   public static void main(String args[]){
     Employee e;
      e= new Manager();
      e.display(); // calls Manager's display()
      e= new Programer();
     e.display();// calls Programer's display()
}}
```

Final Class and Final Method

- **Final class can't be subclass**

```
final class MyClass{----}
```

- **Subclass from MyClass is not possible**
- **class YourClass extends MyClass{ } //compilation error**

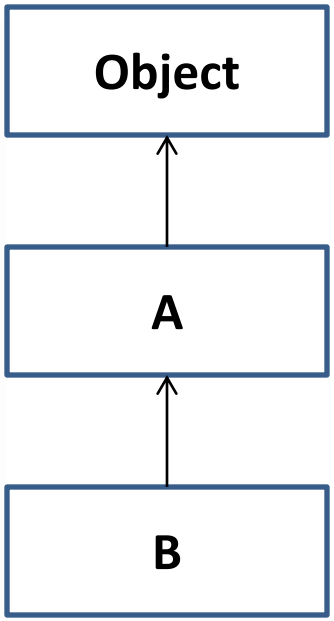
- **Final methods can't be overriden**

```
class XClass{
    final void someMethod(){  ----  }
}
Class YClass extends Xclass{

    void someMethod(){ --- }//compilation error
}
```

The Object Class

- **Object is the base class for all Java classes**
- **Every class extends this class directly or indirectly**
- **Present in the package java.lang which is imported by default into all java programs**



Methods of Object Class

toString()	Returns a string representation of the object
finalize()	Called by the garbage collector on an object when there are no more references to the object
equals()	Indicates if some other object is "equal to" this one
clone()	Creates and returns a copy of this object. The precise meaning of "copy" may depend on the class of the object
hashCode()	Returns a hash code value for the object. This method is supported for the benefit of hashtables such as those provided by java.util.Hashtable

Abstract Class

A class that is declared with abstract keyword

May or may not include abstract methods

- **Abstract methods do not have implementation (body)**
- **Cannot be instantiated, but it can be subclassed**
- **When an abstract class is subclassed, the subclass provides implementations for all abstract methods in its parent class**
- **And, if it does not, the subclass must also be declared abstract**

Abstract Class (Contd…)

```
abstract class Vehicle {
   public abstract void start();
}
```

```
class Car extends Vehicle{
   public void start()
   {
        -------
   }
}
```

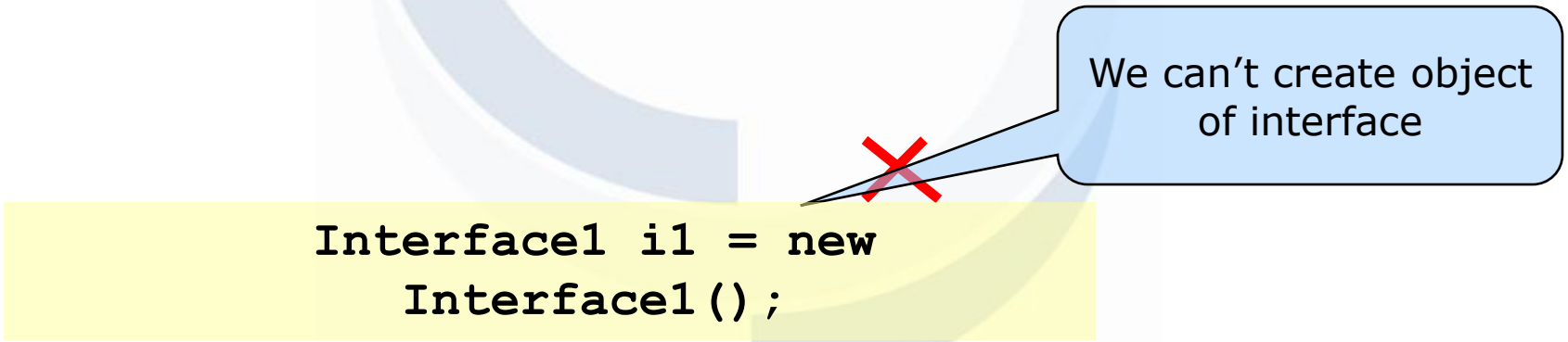
Interface

- In Java, an interface is a reference type, similar to a class
- Can contain constants & method signatures
- There are no method bodies
- Cannot be instantiated — they can only be implemented by classes or extended by other interfaces
- Variables & methods declared in interfaces are public by default
- An interface can extend another interface
- A class can implement multiple interfaces

Interface (Contd…)

Example of an interface declaration:

```
public interface Shape
{   void area();
    void draw();
    void rotate ();
}
```



Interface

```
public class Circle implements Shape
{  private int radius;
    public Circle(){ radius=5; }
    public void draw(){ --- }
    public void  rotate() {  ---- }
    public void area() { ---- }
}
```

```
public class Rectangle implements Shape
{  private int length;
   private int breadth;
    public Rectangle(){ length=5; breadth=5; }
    public void draw(){ --- }
    public void  rotate() {  ---- }
     public void area() { ---- }    }
```

```
public class ShapeApplication
{   public static void main(String args[])
   {    Shape s;
          s= new Circle();
          //calls Circle's methods
          s.area(); s.draw(); s.rotate();
            s= new Rectangle();
            s.area();
            s.draw();  //calls Rectangle's method
            s.rotate();

    }
}
```

```
public class ShapeApplication
{   public static void main(String args[])
   {    Shape s;
          s= new Circle();
          //calls Circle's methods
          s.area(); s.draw(); s.rotate();
            s= new Rectangle();
            s.area();
            s.draw();  //calls Rectangle's method
            s.rotate();

    }
}
```

String Class

- **String class objects are immutable (ie. read only).**
- **String is present *java.lang* pakage**

- **When a change is made to a string, a new object is created and the old one is disused.**
- **This causes extraneous garbage collection if string modifier methods are used too often.**
- **Since strings are stored differently than other data (as a memory address), you can't use the == operator for comparison.**

String Class

- **Methods**

1.	length()	return size of the string
2.	charAt(int index)	return character at the given index
3.	toLowerCase()	converts string to lower case characters
4.	toUpperCase()	converts string to upper case characters
5.	concat(String)	concats two strings
6.	equals(String)	return true if the two strings same
7.	indexOf(char)	returns index of the first occurrence given character
8.	lastIndexOf(char)	returns index of the last occurrence given character
9.	subString(int s,int e)	returns substring from s to e position
10.	compareTo(String)	compares String

Using *StringBuffer* Class

- **Present in *java.lang* package**
- **Unlike class String, StringBuffer represents a string that can be *dynamically modified***
- ***StringBuffer is synchronized***
- **String buffer's capacity can be dynamically increased even though its initial capacity is specified**
- **Should be used while manipulating strings like appending, inserting, and so on**

Using *StringBuffer* Class

- **The constructor has following forms:**
- **StringBuffer() : initializes the buffer size to 16 characters**
- **StringBuffer(int size) : explicitly sets the buffer capacity**
- **StringBuffer(String s): initializes the buffer with contents of s and also reserves another 16 characters for expansion.**

- **Some of the Methods in string buffer class**

Using *StringBuffer* Class

1.	int length()	size of stringbufeer
2.	int capacity()	capacity if stringbuffer
3.	char charAt(int where)	returns char at given position
4.	StringBuffer append(String str)	appends str
5.	StringBuffer append(int num)	appends num
6.	StringBuffer insert(int index,String str)	insert str at index
7.	StringBuffer insert(int index,char ch)	insert ch at index
8.	StringBuffer reverse()	reverse stringbuffer
9.	StringBuffer substring(int start)	returns substring from start
10.	StringBuffer substring(int start.int end)	returns substring from start to end
11.	StringBuffer delete(int start, int end)	deletes substring from start to end

Using *StringBuilder* Class

- **Java StringBuilder class is used to create mutable (modifiable) string.**
- **The Java StringBuilder class is same as StringBuffer class**
- **It is non-synchronized. It is available since JDK 1.5.**
- **Constructor**

1. **StringBuilder(): creates an empty StringBuilder with the initial capacity of 16.**
2. **StringBuilder(String str): creates a string Builder with the specified string.**
3. **StringBuilder(int length): creates an empty string Builder with the specified capacity as length.**

Using *StringBuffer* Class

1.	nt length()	size of stringbufeer
2.	int capacity()	capacity if stringbuffer
3.	char charAt(int where)	returns char at given position
4.	StringBuilder append(String str)	appends str
5.	StringBuilder append(int num)	appends num
6.	StringBuilderinsert(int index,String str)	insert str at index
7.	StringBuffer insert(int index,char ch)	insert ch at index
8.	StringBuilder reverse()	reverse stringbuffer
9.	StringBuilder substring(int start)	returns substring from start
10.	StringBuilder substring(int start.int end)	returns substring from start to end
11.	StringBuilder delete(int start, int end)	deletes substring from start to end

Wrapper Classes

- **Wrapper class in java provides the mechanism *to convert primitive into object and object into primitive*.**
- **Since J2SE 5.0, autoboxing converts primitive into object and object into primitive automatically.**
- **The conversion of primitive into object is known as boxing and**
- **And conversion of object to primitive is unboxing**

Wrapper Class

Primitive Type	Wrapper Class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Wrapper Classes

- **int x=5;**
- **Integer num = new Integer(x);**
- **int y= num.intValue();**
- **Or**
- **Integer num=x;**