HEXAWARE

Core Java

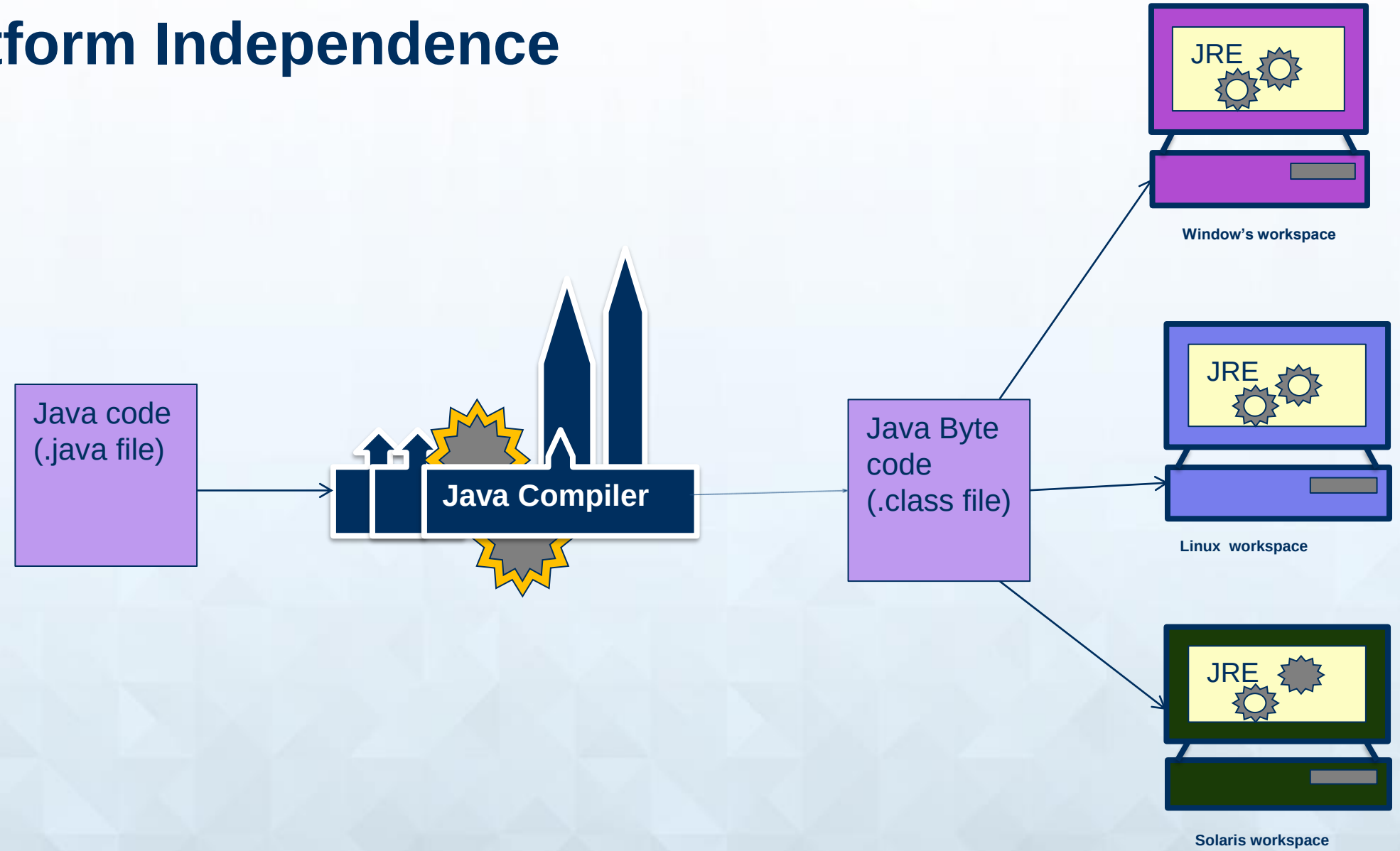
Session Objective

- Uniqueness of Java
- Basics of OOPS
- Variables in Java
- Access specifier and modifiers
- Constructor
- Over Loading and Over riding

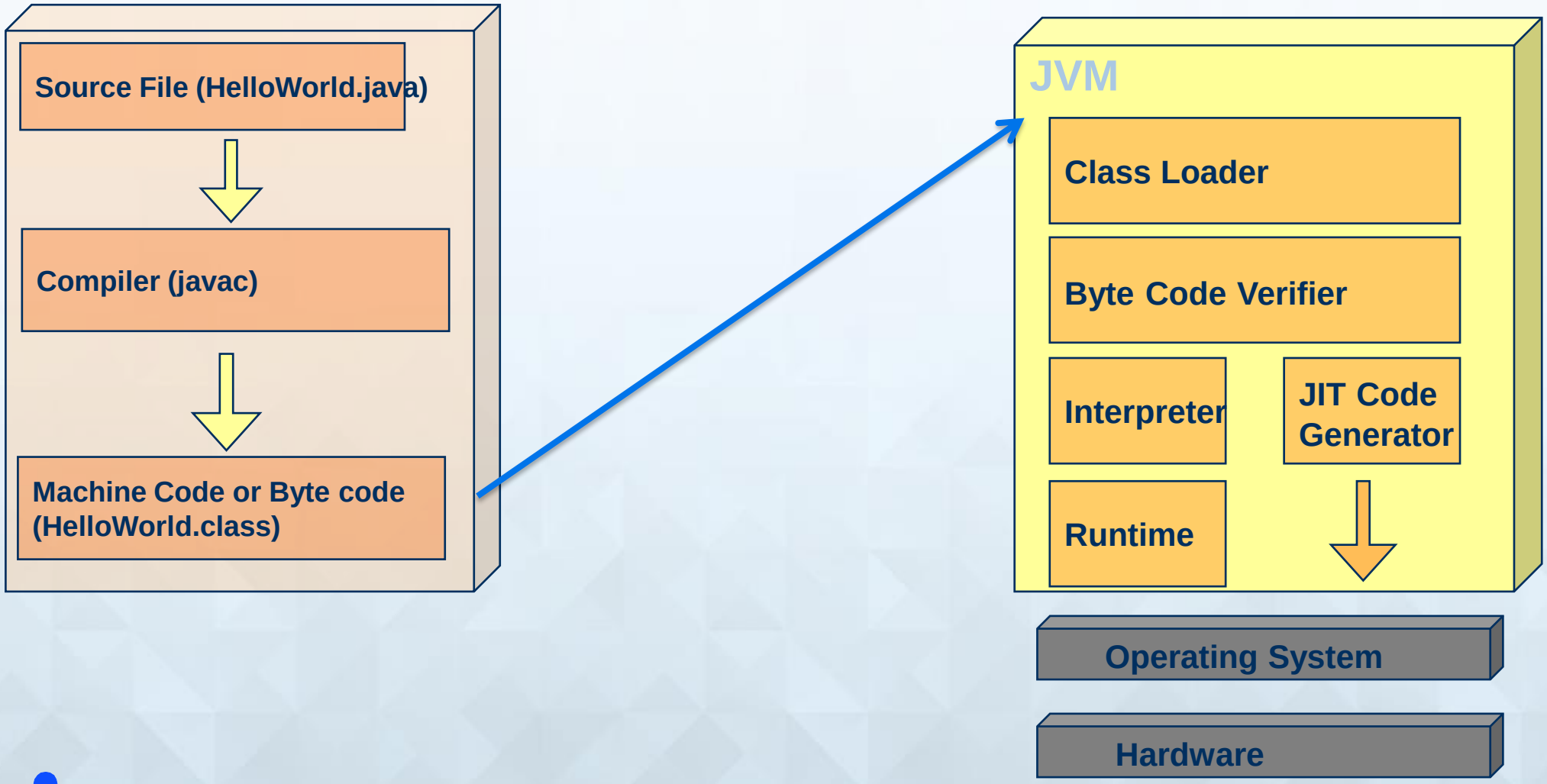
Uniqueness of Java

- Platform Independence
- Robust
- Different Level of Security
- Modular Nature
- Reusable

Platform Independence



Java Architecture:



Gamification - OOPs

OOPS
Gamification

Basics of OOPS

- Encapsulation
- Inheritance
- Polymorphism
- Abstraction

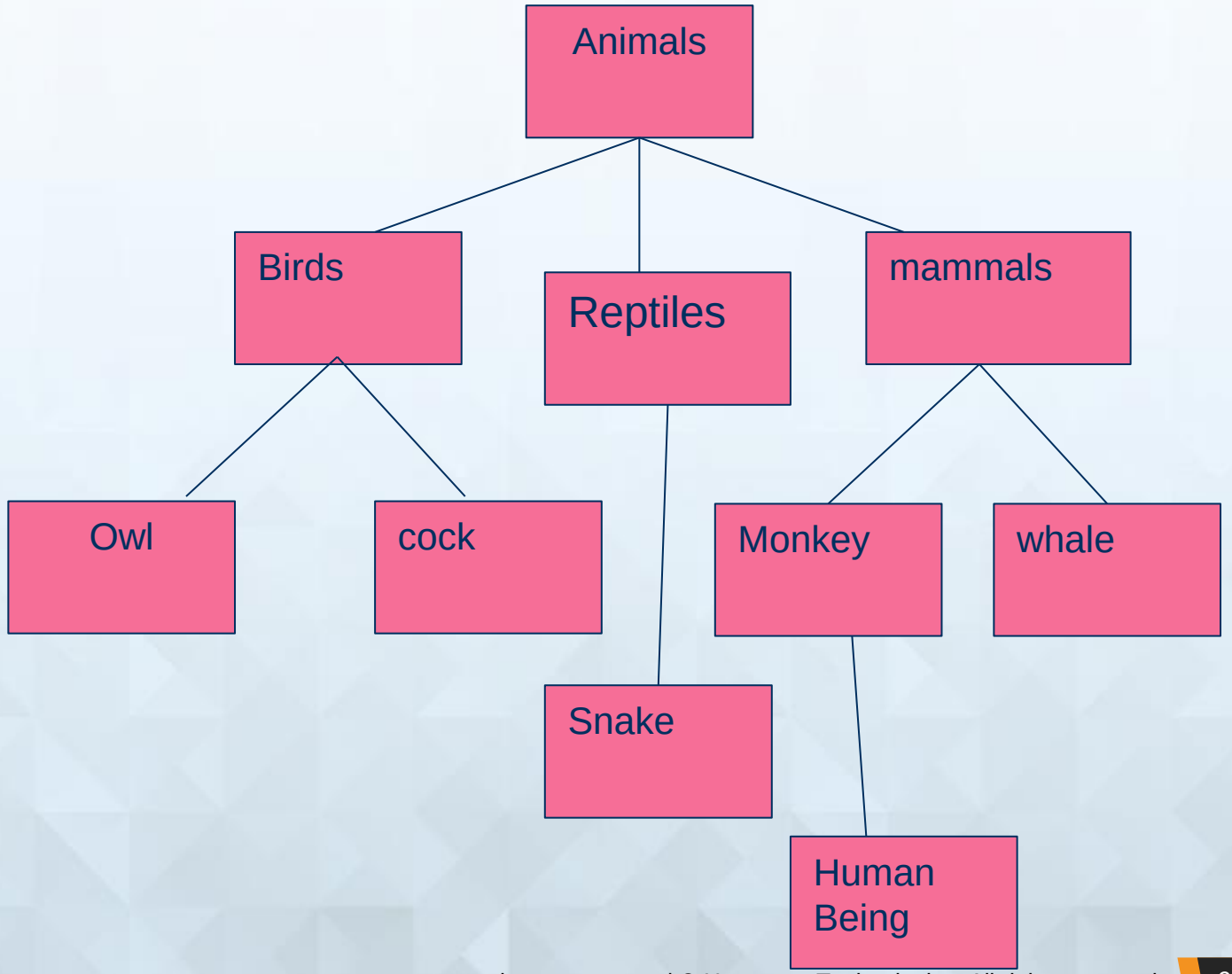
Encapsulation

- **Binding (or wrapping) code and data together into a single unit is known as encapsulation**. For example: capsule, it is wrapped with different medicines.

- A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Inheritance

- Inheritance is a process by which one object acquires the properties of another object.
- An object has to define only those qualities which makes it unique. It can inherit the general attributes from its parent.



Polymorphism

- When one task is performed by different ways i.e. known as polymorphism. For example: to convenes the customer differently, to draw something e.g. shape or rectangle etc.
- In java, we use method overloading and method overriding to achieve polymorphism.
- Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.

Abstraction

- When we drive a car we often need to change the gears of the vehicle but we are not concerned about the inner details of the vehicle engine. What matters to us is that we must shift a gear, that's it. This is abstraction; show only the details that matter to the user.

  Abstraction is used to manage complexity. Software developers use abstraction to decompose complex systems into smaller components

Class and Object cont..

Access specifiers

Java Syntax

```
//creating a class for mobile
class Mobile
{

private int pixels;

private String comp;

private double cost;

public String getModel(int pix,string comp,int cost)

{
return “xyz”;
}
}
```

private int pixels;
private String comp;
private double cost;

3 member variables

Define the class Mobile:
Tell the compiler what the class is made of

public String getModel(int pix,string comp,int cost)
{
return “xyz”;
}

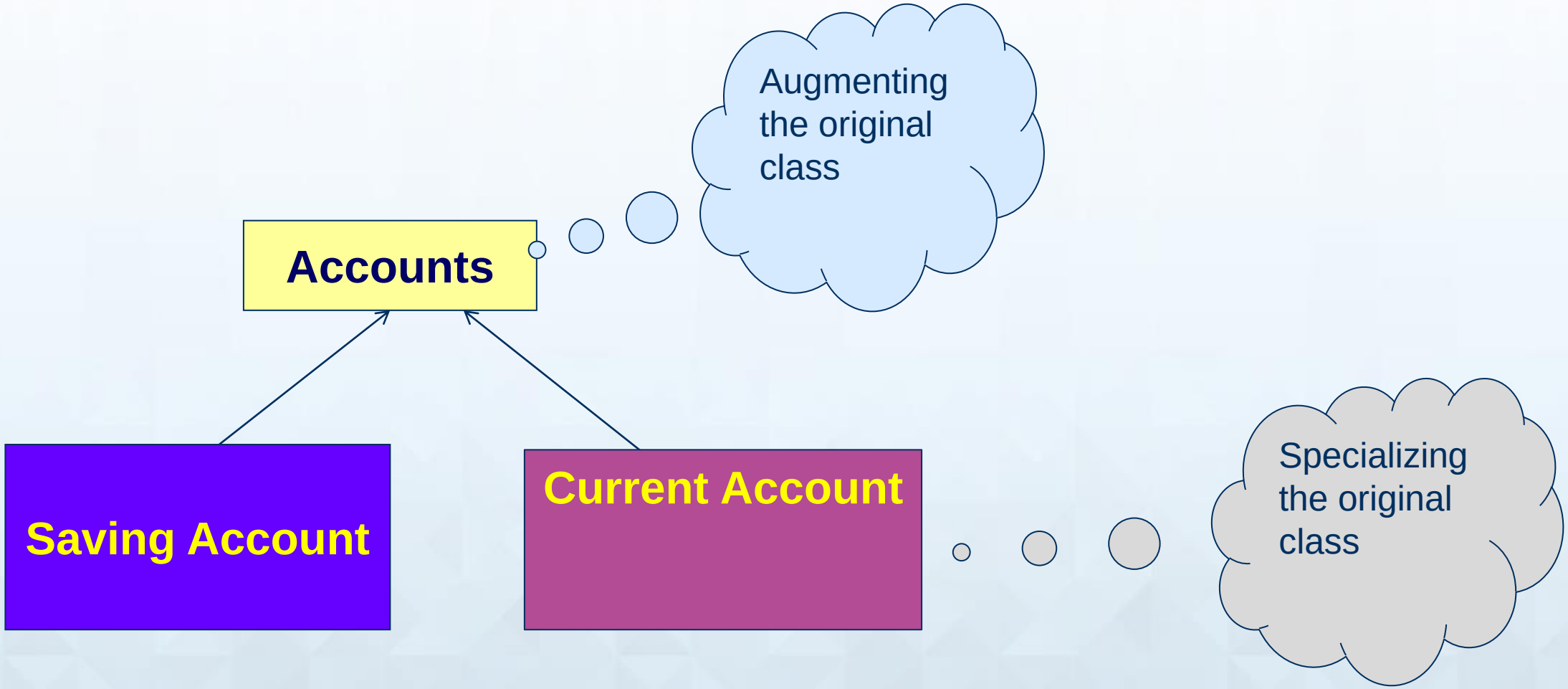
member function

Encapsulation

Encapsulation

```
class Capsule
{
//Outsider don’t have permission to access the private variables
Private String med1;
Private String med2;
//Out siders can call the method as it is declared as public
public String getName(String med1,String med2)
{
String tab=med1+med2+”xyz”;
return tab;
}
```


Inheritance cont..



Inheritance Syntax

```
class Android extends Mobile
{
private String sw;
public String getOs(String sw)
{
Return sw;
}
}
```


Static Polymorphism(compile time polymorphism)

The ability to execute different method implementations by altering the argument used with the method name is known as method overloading.

Overloading

```
Class Banking
{
public String bankATM(String statement)
{
return statement;
}
public String bankATM(int amount,String conf)
{
If(conf.equals(“yes”))
{
return Money;
}
else
return “Transaction failed”;

}
```

```
public String bankATM(String statement)
{
return statement;
}
```

```
public String bankATM(int amount,String conf)
{
If(conf.equals(“yes”))
{
return Money;
}
}
```


Dynamic Polymorphism (run time polymorphism)

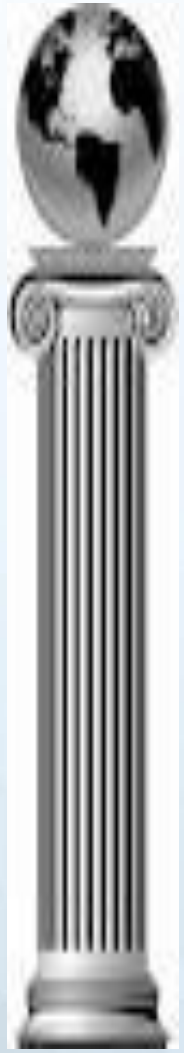
When you create a subclass by extending an existing class, the new subclass contains data and methods that were defined in the original superclass.

In other words, any child class object has all the attributes of its parent. Sometimes, however, the superclass data fields and methods are not entirely appropriate for the subclass objects; in these cases, you want to override the parent class members.

Overriding Syntax

In 1990 model

```
Class Mobile {
String getFunction()
{
Return “a+b+c”;
}
String getName()
{
Return “xyz”;
}
}
```



In 2014 model

```
Class Android extends Mobile
{
String OS()
{
Return “Android”;
}
String getFunction()
{
Return “s+x+n”;
}
}
```


Dynamic Binding

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object

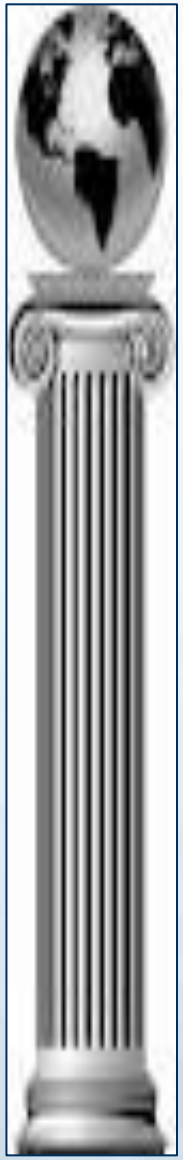
It is important to know that the only possible way to access an object is through a reference variable. A reference variable can be of only one type. Once declared, the type of a reference variable cannot be changed.

The reference variable can be reassigned to other objects provided that it is not declared final. The type of the reference variable would determine the methods that it can invoke on the object.

A reference variable can refer to any object of its declared type or any subtype of its declared type. A reference variable can be declared as a class or interface type.

Dynamic Binding cont…

```
Class SolarLight
{
String gives()
{
Return “light till 300 mts”;
}
String gives1()
{
Return “heat till 50 mts”;
}
public String Energy()
{
        Return “traditional energy”;
        }
}
```



```
Class Sun extends SolarLight
{       public String Energy()
{
        Return solar energy;
        }
public String Gasses()
{
        String gases= “Carbon dioxide (CO2)
        ,Methane (CH4) “;
        return gases
}
public String rays()
{
        String ray=” infrared ,ultraviolet rays “;
        return ray;
}
}
```

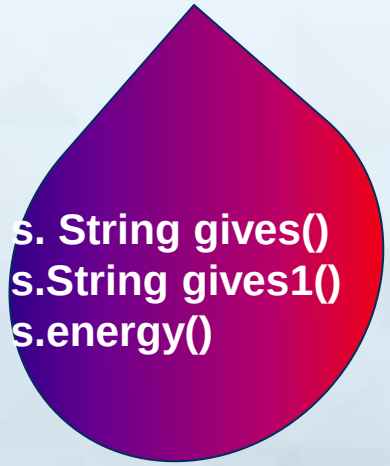

Dynamic Binding cont…

```
Class Myclass
{
Public static void main(String ar[])
{
//create reference for class solarlight
Solarlight s=null;
//creating object for sun
S=new Sun();
//using s we can get all the methods from solarlight and only the overridden methods from sun
s. String gives()       //ans is (“light till 300 mts”)
 S.String gives1()     //ans is (Return “heat till 50 mts”)
//ans is (Return solar energy)
 s. String gives()
S.String gives1()
S.energy()

}
}
```

Solarlight s=null;

s=new Sun;



OOPS – Case study

OOPS_CaseStudy

Access specifier

- An access specifier is a defining code element that can determine which elements of a program are allowed to access a specific variable or other piece of data.

Access Specifiers
public
private
protected
package

Access modifiers

We know access specifiers specify the access of variable ,methods and classes and access modifiers modify the access of variables, methods and classes. In general, an access modifier works between the classes of the same application or within the classes of same package.

Access Modifiers
Final
Static
Transient
Volatile
synchronized

Variables in Java

- Local Variables
- Instance Variable
- Class Variable
- Reference Variable

Static & Final

Static

- Static variable
- Static method
- Static block

Final

- Final variable
- Final method
- Final class

Constructor

- A java constructor has the same name as the name .
- No return type is there for a constructor.
- Constructors can be parameterized .
- Java provides a default constructor when no explicit constructors are provided.
- Super is a keyword used to call parent constructor.
- This is a key word used to call same class constructor.

Constructor

Constructor is a special type of method that is used to initialize the object.

Constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor.

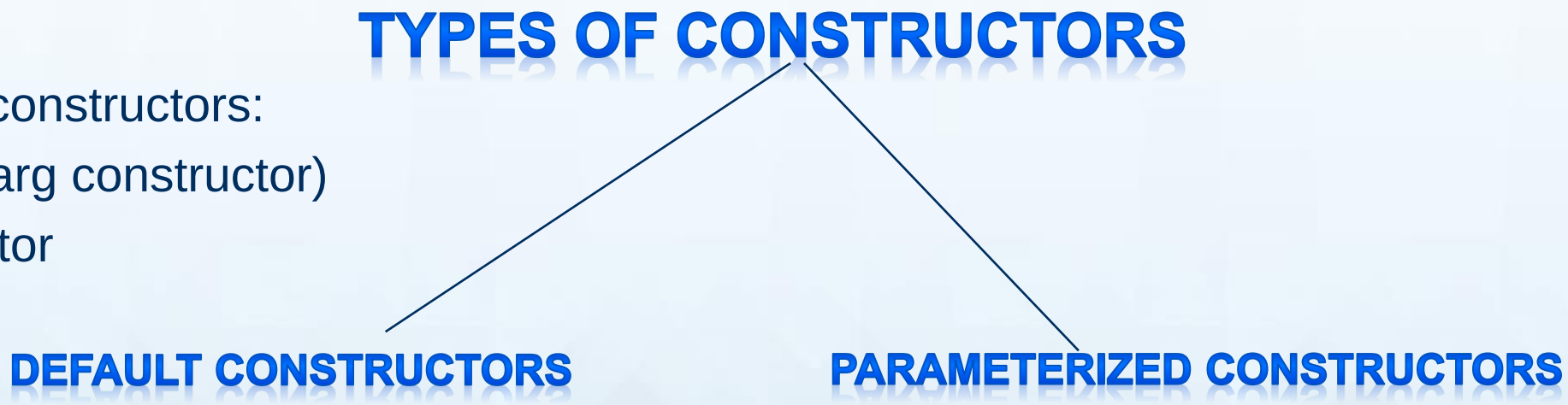
Rules for creating constructor

- There are basically two rules defined for the constructor.
- Constructor name must be same as its class name
- Constructor must have no explicit return type

Constructor cont..

Types of constructors

There are two types of constructors:
default constructor (no-arg constructor)
parameterized constructor



Constructor cont….

CAN WE OVER LOAD A CONSTRUCTOR

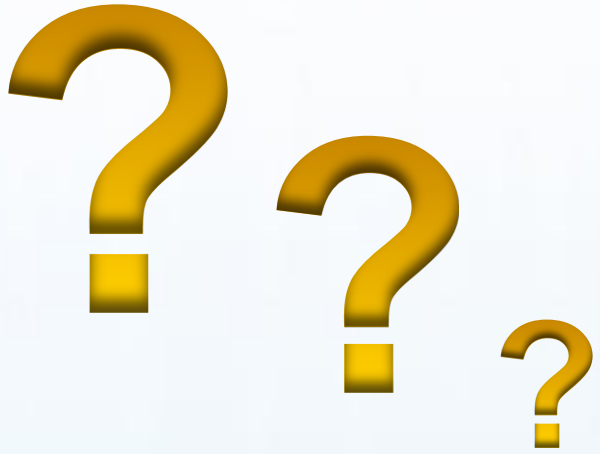
Over Loading

- A class have multiple methods by same name but different parameters, it is known as Method Overloading.
- If we have to perform only one operation, having same name of the methods increases the readability of the program.
- Advantage of method overloading

  Method overloading simplify and increases the readability of the program.
- Different ways to overload the method

  By changing number of arguments

  By changing the data type



Is it possible to over ride a main method

Over Riding

Over Riding :

If subclass has the same method as declared in the parent class, it is known as **method overriding**.

In other words, If subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as Method Overriding.

Advantage of Java Method Overriding :

Method Overriding is used to provide specific implementation of a method that is already provided by its super class.

Method Overriding is used for Runtime Polymorphism

Rules for Method Overriding :

method must have same name as in the parent class

method must have same parameter as in the parent class.

must be IS-A relationship (inheritance).

Abstract class

Java Abstract classes are used to declare common characteristics of subclasses.

An abstract class cannot be instantiated. It can only be used as a superclass for other classes that extend the abstract class.

Abstract classes are declared with the abstract keyword.

Abstract classes are used to provide a template or design for concrete subclasses down the inheritance tree.

Like any other class, an abstract class can contain fields that describe the characteristics and methods that describe the actions that a class can perform.

An abstract class can include methods that contain no implementation.

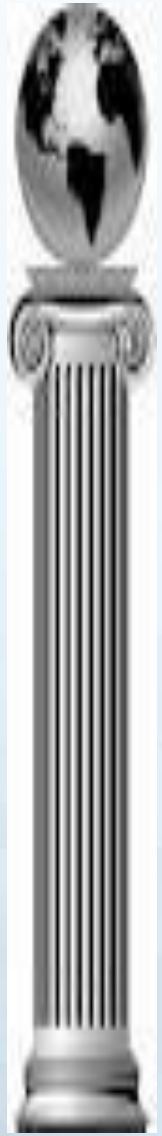
These are called abstract methods.

Abstract classes cannot be instantiated

They must be sub classed, and actual implementations must be provided for the abstract methods.

Abstract class cont…

```
public abstract class Vehicle
{
        abstract boolean hasDiskBrake();
        abstract int getNoofGears();
// method with implementation (concreate method)
        String ownerShip(String own)
        {
                Return own;;
        }
}
```



```
Class Car extends Vehicle
{
//overriding the abstract methods
        boolean hasDiskBrake()
        {
                Return true;
        }
         int getNoofGears()
        {
                Return 2;
        }
 //implementing its own logic
        String color()
        {
                Return “red”;
        }
}
```

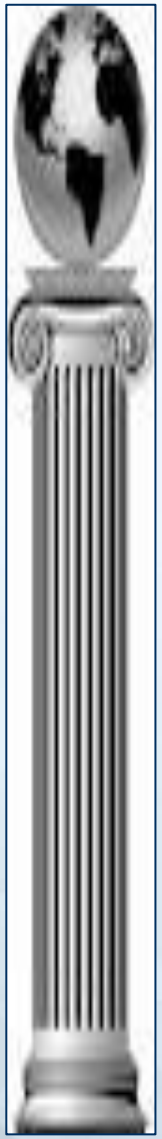

Interface

- An Interface in java is known as Contract which must be followed by its implementing class (Child class).
- Only public and abstract modifiers are allowed to use in interface. (Static are not allowed)
- If variable is defined inside the interface will be considered as final by default.
- An interface cannot be instantiated directly.
- A class can implement more than one interface.
- An interface itself can inherit from multiple interfaces.
- The interface itself contains no implementations but only the public members signature

Interface cont…

```
interface Shape {

        public double area();
        public double volume();
}
```



```
public class Point implements Shape {
        static int x, y;
        public Point() {
                x = 0;
                y = 0;
        }
        public double area() {
                return 2000;
        }
        public double volume() {
                return 7878;
        }
        public static void print() {
        System.out.println("point: " + x + "," + y);
        }
        public static void main(String args[]) {
        Point p = new Point();
        p.print();
}
```


Abstract class Vs Interface

Interface	Abstract class
Interface are implicitly abstract and cannot have implementations.	An abstract class can have instance methods that implements a default behavior.
Variables declared in a Java interface is by default final.	An abstract class may contain non-final variables.
Members of a Java interface are public by default.	A Java abstract class can have the usual flavors of class members like private, protected, etc..
Java interface should be implemented using keyword “implements”; A Java abstract class should be extended using keyword “extends”.	An interface can extend another Java interface only, an abstract class can extend another Java class and implement multiple Java interfaces.
A Java class can implement multiple interfaces but it can extend only one abstract class.	Abstract class can extends only one class
Interface is absolutely abstract and cannot be instantiated	A Java abstract class also cannot be instantiated

Inner Classes

- In java we can write a class inside the class. These types of classes called Inner classes, there are four type of inner classes which you can write in java classes

1. **Static Inner Classes**
2. **Member Inner Class**
3. **Method-Local Inner Classes**
4. **Anonymous Inner Classes**

Static Inner Classes

- In static class we cannot declare static members like variables or methods or static block.
- In Inner class we can access only static member of outer class.
- If you declare inner class as static. The class is called as static inner class.

Static Inner Classes

```
class Outer
 {
      static class Inner
       {
            void go()
                {
              System.out.println("Inner class reference is: " + this);
            }
       }
}

public class Test {
            public static void main(String[] args)
            {
                        Outer.Inner n = new Outer.Inner();
                        n.go();
            }
```

Test.java

OOPS - Assignment

OOPS
ASsignment

HEXAWARE

Innovative Services

Passionate Employees

Delighted Customers

Thank you

www.hexaware.com