JAVA OOPS Concepts

INSV 20K
10
72
INSV 40K
12

Class and Object

Class: blueprint
object: inherits Class

```
public class MyClass {
  int x = 5;

  public static void main(String[] args) {
    MyClass myObj = new MyClass();
    System.out.println(myObj.x);
  }
}
```

class should always start with an uppercase first letter, and that the name of the java file should match the class name.

Compilation

```
C:\Users\Name>javac MyClass.java
C:\Users\Name>javac OtherClass.java.
C:\Users\Name>java OtherClass
```

Constructors

*Note that the constructor name must **match the class name**, and it cannot have **a return type** (like void).*

```
public MyClass() {
   x = 5;  // Initial Value of x
 }
```

Inner Classes

Possible to make nested inner classes

```
class OuterClass {
 int x = 10;
 private class InnerClass {
  int y = 5;
 }
}
```

This supports modifiers too

Iterface ✓

- An interface is a completely "**abstract class**" that is used to group related methods with empty bodies

```
interface Animal {
    public void animalSound();
// empty body
    public void run();
// empty body
}
```

- Java does not support "multiple inheritance" (a class can only inherit from one superclass). However, it can be achieved with interfaces, because the class can **implement** multiple interfaces. ***Note:*** *To implement multiple interfaces, separate them with a comma*

```
interface FirstInterface {
  public void myMethod(); // interface method
}

interface SecondInterface {
  public void myOtherMethod(); // interface method
}

class DemoClass implements FirstInterface, SecondInterface {
  public void myMethod() {
    System.out.println("Some text..");
  }
  public void myOtherMethod() {
    System.out.println("Some other text...");
  }
}
```

Static V/s Non Static

A static method, which means that it can be accessed without creating an object of the class, unlike public, which can only be accessed by objects:

```
public class MyClass {
    // Static method
    static void myStaticMethod() {
      System.out.println("Static methods can be called without creating objects");
    }

    // Public method
    public void myPublicMethod() {
      System.out.println("Public methods must be called by creating objects");
    }

    // Main method
    public static void main(String[] args) {
      myStaticMethod(); // Call the static method
      // myPublicMethod(); This would compile an error

      MyClass myObj = new MyClass(); // Create an object of MyClass
      myObj.myPublicMethod(); // Call the public method on the object
    }
  }
```

Encapsulation

The meaning of **Encapsulation**, is to make sure that "sensitive" data is hidden from users. To achieve this, you must:

- declare class variables/attributes as private
- provide public **get** and **set** methods to access and update the value of a private variable

private variables can only be accessed within the same class (an outside class has no access to it). However, it is possible to access them if we provide public **get** and **set** methods.

```
public class Person {
    private String name; // private = restricted access
    // Getter
    public String getName() {
        return name;
    }
    // Setter
    public void setName(String newName) {
        this.name = newName;
    }
}
public class MyClass {
    public static void main(String[] args) {
        Person myObj = new Person();
        myObj.name = "John";  // error
        System.out.println(myObj.name); // error
        myObj.setName("John"); // Set the value of the name
        System.out.println(myObj.getName());
    }
```

Polymorphism

- Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.
- [Inheritance](#) lets us inherit attributes and methods from another class. Polymorphism uses those methods to perform different tasks.

Vehicle to a protected [access modifier](#). If it was set to private, the Car class would not be able to access it. If you try to access a final class, Java will generate an error

```
class Animal {
    public void animalSound() {
        System.out.println("The animal makes a sound");
    }
}
class Pig extends Animal {
    public void animalSound() {
        System.out.println("The pig says: wee wee");
    }
}
class Dog extends Animal {
    public void animalSound() {
        System.out.println("The dog says: bow wow");
    }
}
class MyMainClass {
    public static void main(String[] args) {
        Animal myAnimal = new Animal();  // Create a An
        Animal myPig = new Pig();  // Create a Pig obje
        Animal myDog = new Dog();  // Create a Dog obje
        myAnimal.animalSound();
        myPig.animalSound();
        myDog.animalSound();
    }
}
```

Modifiers(Attributes and methods)

Modifier	Description
final	Attributes and methods cannot be overridden/modified
static	Attributes and methods belongs to the class, rather than an object
abstract	Can only be used in an abstract class, and can only be used on methods. The method does not have a body, for example abstract void run();. The body is provided by the subclass (inherited from).
transient	Attributes and methods are skipped when serializing the object containing them
synchronized	Methods can only be accessed by one thread at a time
volatile	The value of an attribute is not cached thread-locally, and is always read from the "main memory"

Inheritance

- **subclass** (child) - the class that inherits from another class
- **superclass** (parent) - the class being inherited from

To inherit from a class we use the extends keyword.

```
class Vehicle {
    protected String brand = "Ford";        // Vehicle attribute
    public void honk() {                    // Vehicle method
        System.out.println("Tuut, tuut!");
    }
}

class Car extends Vehicle {
    private String modelName = "Mustang";    // Car attribute
    public static void main(String[] args) {
        // Create a myCar object
        Car myCar = new Car();
        // Call the honk() method (from the Vehicle class) on the myCar object
        myCar.honk();
        // Display the value of the brand attribute (from the Vehicle class)
        // and the value of the modelName from the Car class
        System.out.println(myCar.brand + " " + myCar.modelName);
    }
}
```

Vehicle to a protected [access modifier](#). If it was set to private, the Car class would not be able to access it. If you try to access a final class, Java will generate an error

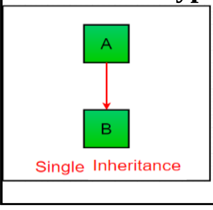
Abstraction

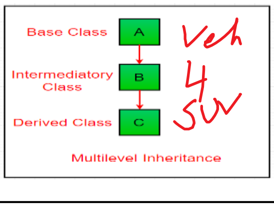
Data **abstraction** is the process of hiding certain details and showing only essential information to the user.
The abstract keyword is a non-access modifier, used for classes and methods

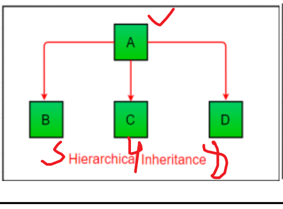
- **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
- **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

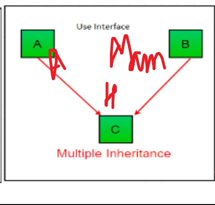
```
abstract class Animal {
    // Abstract method (does not have a body)
    public abstract void animalSound();
    // Regular method
    public void sleep() {
        System.out.println("Zzz");
    }
}
Animal myObj = new Animal(); // will generate an error

// Subclass (inherit from Animal)
class Pig extends Animal {
    public void animalSound() {
        // The body of animalSound() is provided here
        System.out.println("The pig says: wee wee");
    }
}
class MyMainClass {
    public static void main(String[] args) {
        Pig myPig = new Pig(); // Create a Pig object
        myPig.animalSound();
        myPig.sleep();
    }
}
```

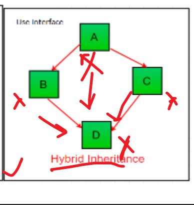
Inheritance Types in JAVA



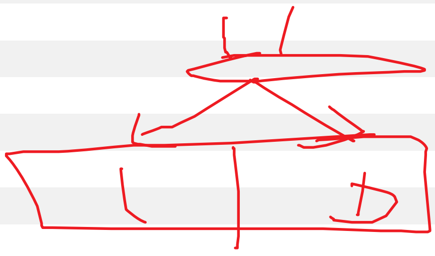




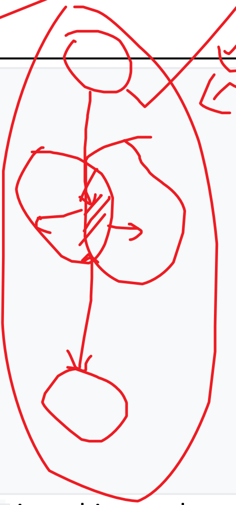




Keyword	Description
abstract	A non-access modifier. Used for classes and methods: An abstract class cannot be used to create objects (to access it, it must be inherited from another class). An abstract method can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from)
assert	For debugging
boolean	A data type that can only store true and false values
break	Breaks out of a loop or a switch block
byte	A data type that can store whole numbers from -128 and 127
case	Marks a block of code in switch statements
catch	Catches exceptions generated by try statements
char	A data type that is used to store a single character
class	Defines a class
continue	Continues to the next iteration of a loop
const	Defines a constant. Not in use - use final instead
default	Specifies the default block of code in a switch statement
do	Used together with while to create a do-while loop
double	A data type that can store whole numbers from 1.7e−308 to 1.7e+308
else	Used in conditional statements
enum	Declares an enumerated (unchangeable) type
exports	Exports a package with a module. New in Java 9
extends	Extends a class (indicates that a class is inherited from another class)
final	A non-access modifier used for classes, attributes and methods, which makes them non-changeable (impossible to inherit or override)
finally	Used with exceptions, a block of code that will be executed no matter if there is an exception or not
float	A data type that can store whole numbers from 3.4e−038 to 3.4e+038
for	Create a for loop
goto	Not in use, and has no function
if	Makes a conditional statement
implements	Implements an interface
import	Used to import a package, class or interface
instanceof	Checks whether an object is an instance of a specific class or an interface
int	A data type that can store whole numbers from -2147483648 to 2147483647
interface	Used to declare a special type of class that only contains abstract methods
long	A data type that can store whole numbers from -9223372036854775808 to 9223372036854775808
module	Declares a module. New in Java 9
native	Specifies that a method is not implemented in the same Java source file (but in another language)
new	Creates new objects
package	Declares a package
private	An access modifier used for attributes, methods and constructors, making them only accessible within the declared class
protected	An access modifier used for attributes, methods and constructors, making them accessible in the same package and subclasses
public	An access modifier used for classes, attributes, methods and constructors, making them accessible by any other class
requires	Specifies required libraries inside a module. New in Java 9
return	Finished the execution of a method, and can be used to return a value from a method
short	A data type that can store whole numbers from -32768 to 32767
static	A non-access modifier used for methods and attributes. Static methods/attributes can be accessed without creating an object of a class
strictfp	Restrict the precision and rounding of floating point calculations
super	Refers to superclass (parent) objects
switch	Selects one of many code blocks to be executed
synchronized	A non-access modifier, which specifies that methods can only be accessed by one thread at a time
this	Refers to the current object in a method or constructor
throw	Creates a custom error
throws	Indicates what exceptions may be thrown by a method
transient	A non-accesss modifier, which specifies that an attribute is not part of an object's persistent state
try	Creates a try...catch statement
var	Declares a variable. New in Java 10
void	Specifies that a method should not have a return value
volatile	Indicates that an attribute is not cached thread-locally, and is always read from the "main memory"
while	Creates a while loop



Virtual in C++



```
struct Animal {
    virtual ~Animal() = default;
    virtual void Eat() {}
};

struct Mammal: Animal {
    virtual void Breathe() {}
};

struct WingedAnimal: Animal {
    virtual void Flap() {}
};

// A bat is a winged mammal
struct Bat: Mammal, WingedAnimal {};

Bat bat;
```

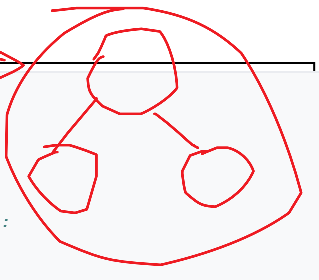
As declared above, a call to `bat.Eat` is ambiguous because there are two `Animal` (indirect) base classes in `Bat`, so any `Bat` object has two different `Animal` base class subobjects. There is no way computer can disambiguate about what eat you're calling hence

```
Bat b;
Animal& a = b;  // error: which Animal subobject should a Bat cast into,
                // a Mammal::Animal or a WingedAnimal::Animal?
```

To disambiguate, one would have to explicitly convert `bat` to either base class subobject:

```
Bat b;
Animal& mammal = static_cast<Mammal&>(b);
Animal& winged = static_cast<WingedAnimal&>(b);
```

In order to call `Eat`, the same disambiguation, or explicit qualification is needed.



```
struct Animal {
    virtual ~Animal() = default;
    virtual void Eat() {}
};

// Two classes virtually inheriting Animal:
struct Mammal: virtual Animal {
    virtual void Breathe() {}
};

struct WingedAnimal: virtual Animal {
    virtual void Flap() {}
};

// A bat is still a winged mammal
struct Bat: Mammal, WingedAnimal {};
```

The `Animal` portion of `Bat::WingedAnimal` is now the *same* `Animal` instance as the one used by `Bat::Mammal`, which is to say that a `Bat` has only one, shared, `Animal` instance in its representation and so a call to `Bat::Eat` is unambiguous. Additionally, a direct cast from `Bat` to `Animal` is also unambiguous, now that there exists only one `Animal` instance which `Bat` could be converted to.

The ability to share a single instance of the `Animal` parent between `Mammal` and `WingedAnimal` is enabled by recording the memory offset between the `Mammal` or `WingedAnimal` members and those of the base `Animal` within the derived class. However this offset can in the general case only be known at runtime, thus `Bat` must become (`vpointer`, `Mammal`, `vpointer`, `WingedAnimal`, `Bat`, `Animal`). There are two vtable pointers, one per inheritance hierarchy that virtually inherits `Animal`. Increases memory footprint but solves the problem

What are manipulators?
Manipulators are the functions which can be used in conjunction with the insertion (<<) and extraction (>>) operators on an object. Examples are endl and setw.

What is a virtual function?
A virtual function is a member function of a class, and its functionality can be overridden in its derived class. This function can be implemented by using a keyword called virtual, and it can be given during function declaration. A virtual function can be declared using a token(virtual) in C++. It can be achieved in C/Python Language by using function pointers or pointers to function.

What are tokens?
A compiler recognizes a token, and it cannot be broken down into component elements. Keywords, identifiers, constants, string literals, and operators are examples of tokens. Even punctuation characters are also considered as tokens. Example: Brackets, Commas, Braces, and Parentheses.

What is static and dynamic Binding?
Binding is nothing but the association of a name with the class. Static Binding is a binding in which name can be associated with the class during compilation time, and it is also called as early Binding.
Dynamic Binding is a binding in which name can be associated with the class during execution time, and it is also called as Late Binding.

What do you mean by finally block?
A finally block consists of a system that is used to perform significant code such as closing a connection, etc. This block performs when the try block exits. It also makes sure that lastly, block executes even in case some unforeseen exception is encountered.

Can you explain what operator overloading is?
The term operator overloading means that depending on the arguments passed, the operators' behaviour can be changed. However, it works only for user-defined types.

What is a friend function?
A friend function is a friend of a class that is allowed to access to Public, private, or protected data in that same class. If the function is defined outside the class cannot access such information.
A friend can be declared anywhere in the class declaration, and it cannot be affected by access control keywords like private, public, or protected.

What is a ternary operator?
The ternary operator is said to be an operator which takes three arguments. ? :

What is the super keyword?
The super keyword is used to invoke the overridden method, which overrides one of its superclass methods. This keyword allows to access overridden methods and also to access hidden members of the superclass. It also forwards a call from a constructor, to a constructor in the superclass.

What is the main difference between overloading and overriding?
Overloading is static Binding, whereas Overriding is dynamic Binding. Overloading is nothing but the same method with different arguments, and it may or may not return the equal value in the same class itself. Overriding is the same method names with the same arguments and return types associated with the class and its child class. It is mainly inherited in Nature.

What is a copy constructor?
This is a special constructor for creating a new object as a copy of an existing object. There will always be only one copy constructor that can be either defined by the user or the system.

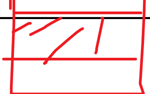
Define Garbage collection?
GC is an implementation of automatic memory management. The Garbage collector liberated up space engaged by objects that are no longer in existence.

Define manipulators?
Manipulators are the functions which can be used in combination with the placing (<<) and withdrawal (>>) operators on an object. Examples: end and set.

What are sealed modifiers?
Sealed modifiers are the access modifiers where the systems cannot inherit it. Sealed modifiers can also be functional to properties, events, and methods. This modifier cannot be used to static members.

Dynamic Dispatch?
Dynamic dispatch also known as message passing is a process of selecting a procedure to run in response to a method call by looking for the method (function) in the table associated with the object, at run time. It distinguishes an object from a module which has fixed implementations for all instances i.e. static dispatch. For example there are three classes. Class X- the base class, Class Y and Class Z- the derived class and all of them have a function call- show( ), then dynamic dispatch selects which implementation of function to call at run time.

python saves the name and variables of Class during execture



before running the code C++ saves the name of the classes in the memory