

Object Oriented Principles

A large, faint, light-gray watermark of the word "Enlighters" is centered in the background. It includes a lightbulb icon above the 'i' and a graduation cap icon above the 's'.

Identifiers & Data Types

Identifiers ??

- All Java components require names. Name used for classes, methods, interfaces and variables are called Identifier.
- A variable takes up a certain amount of space in memory. How much memory a variable takes depends on its data type
- Identifier must follow some rules. Here are the rules:
 - All identifiers must start with either a letter(a to z or A to Z) or currency character(\$) or an underscore.
 - After the first character, an identifier can have any combination of valid characters (**currency character, letter, underscore or digits**)
 - A **Java keyword** cannot be used as an identifier
 - Identifiers in Java are case sensitive, foo and Foo are two different identifiers
 - Cannot also use "true, false and null" as identifiers as they are literals
 - **Whitespaces** are not allowed

Identify...

```

aA      Baba      abracrbb
bbb@    #heello    class  classes
      AbrAcAdAbrA  identifier-main abra_cadabra      abra$      abra1
$abra$ _abra  ____ab 123abc
Abc123 float  null
  
```

Data Types in Java

- Java is a strongly typed language
- Variables must be declared before use and each variable takes a certain amount of space in memory depending on the data type
- Data Types are of two groups:
 - Primitive Data Types
 - Object References
- Primitive Data Types contains the value of the variable directly in the memory allocated to the variable
- Object References will not contain the object itself instead will be holding the memory location where the whole object is stored. It is possible to have many different variables reference the same object.

Primitive Data Types

Data Type	Description
boolean	A binary value of either true or false
byte	8 bit signed value, values from -128 to 127
short	16 bit signed value, values from -32.768 to 32.767
char	16 bit Unicode character
int	32 bit signed value, values from -2.147.483.648 to 2.147.483.647
long	64 bit signed value, values from -9.223.372.036.854.775.808 to 9.223.372.036.854.775.808
float	32 bit floating point value
double	64 bit floating point value

Wrappers

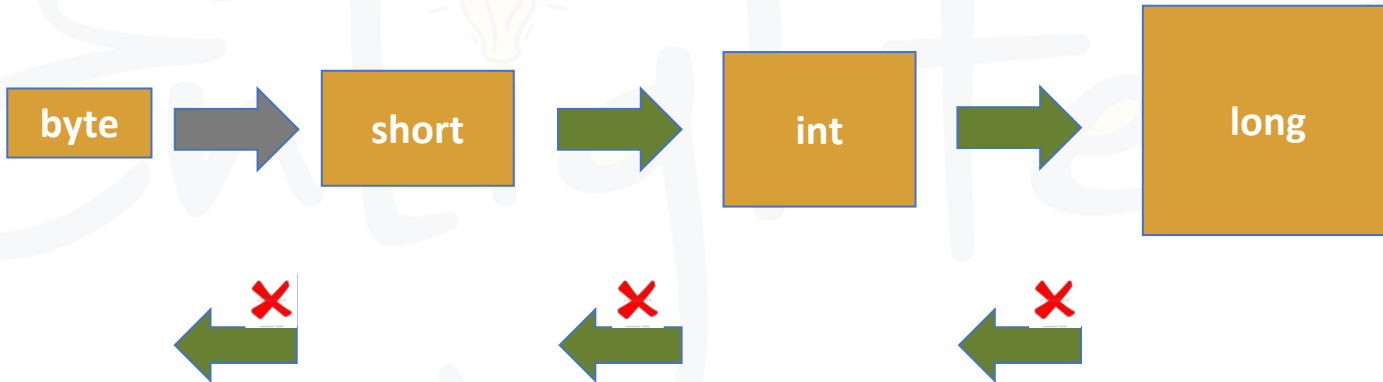
- For each Primitive types, there is a corresponding object version defined in Java called as Wrappers Object
- Enables having multiple references to the same value
- Objects of Primitive types can be identified by a capital letter in the beginning of their name and also by abbreviation differences

Integer sampleIntObj = new Integer(45)

- The object versions of the primitive data types are immutable
- It provides methods for converting between primitive type and strings

Conversions & Casts...

- Java **automatically** converts a lower data type value to a larger type



- Needs a cast to convert from larger type to smaller type but may get the value truncated based on new type

Auto Boxing

- From Java 5 you have a concept called "auto boxing". That means that Java can automatically "box" a primitive variable in an object version, if that is required, or "unbox" an object version of the primitive data type if required.











```
Integer myInteger = new Integer(45);
```

```
int myInt = myInteger;
```

```
int anotherInt = 45;
```

```
Integer intObject = anotherInt;
```

Let's test who was sleeping

- `int X = 3;` 
- `char char123 = "char";` 
- `Integer intVal5 = 5;` 
- `Integer intObj46 = new Integer(46);` 
- `String str = "Hello World \n";` 
- `int val89 = 89; long longVal89 = val89;` 
- `Long longVal10 = 1000004L; int intVal = longVal10;` 
- `Long longVal = 10L; int intVal = (Integer)longVal;` 
- `long longVal33 = 33; int intVal33 = (int)longVal33;` 
- `intX = 10;` 

Java Comments

Comments

```
/*
```

This is a multiline comment. It may carry over many lines, until a closing *asterisk slash* */

```
// This is a single-line comment
```

```
/**
```

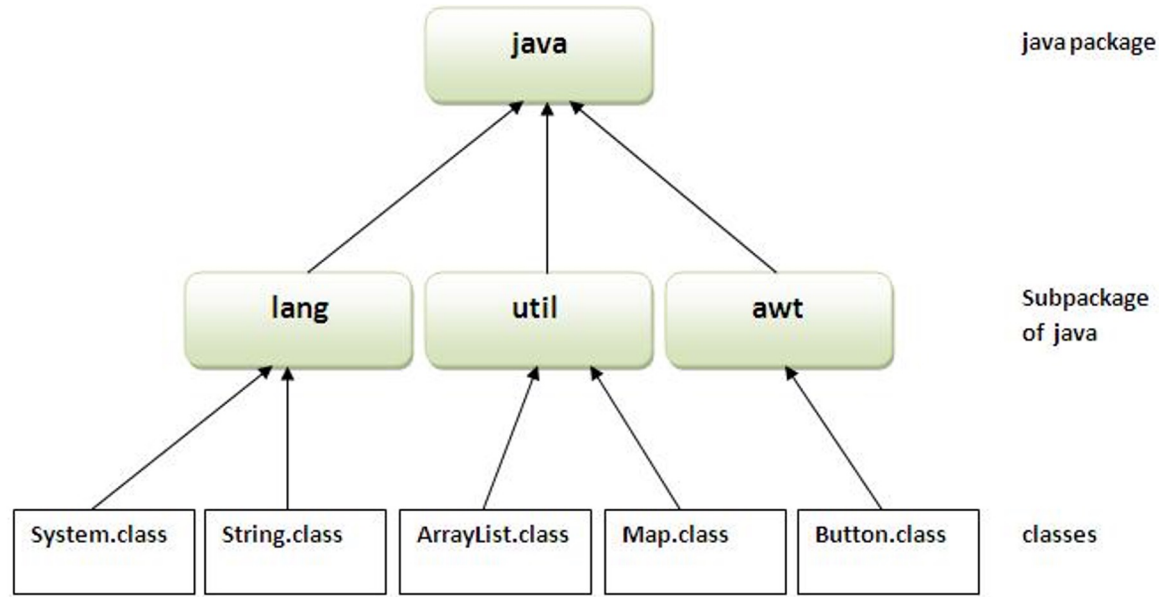
This Doc comment is used by the *javadoc* utility to produce documentation

```
*/
```

Package & Imports

- Java package is a group of similar types of classes, interfaces and sub-packages
- Only one package statement but can have multiple imports
- The package statement (if any) must be the first (non-comment) line in a java file followed by imports before the class declaration
- Why packaging so important?
 - categorize the classes and interfaces so that they can be easily maintained
 - Provides access protection
 - Removes naming collision

Package & Imports



Package & Imports exercise...

- Perform an addition operation from Sample Class by invoking utility API in another package and print the results

Class Access Modifiers...

Access modifiers are used to control the visibility of a class or a variable or a method or a constructor

Access Modifiers	Inside class	Same package class	Same package Sub-Class	Other Package Class	Other Package Sub-Class
public	Y	Y	Y	Y	Y
default/no-access	Y	Y	Y	N	N
protected	Y	Y	Y	N	Y
private	Y	N	N	N	N

Class Non Access Modifiers...

- Classes can also be modified with final, abstract, or strictfp
- Class cannot be both final and abstract
- final class cannot be sub-classed
- An abstract class cannot be instantiated
- A single abstract method in a class means the whole class must be abstract
- An abstract class can have both abstract and non-abstract methods
- The first concrete class to extend an abstract class must implement all of its abstract methods

Access vs Non Access Modifiers

Access Modifiers	Non-Access Modifiers
<p>Private</p> <p>Default or no access modifiers</p> <p>Protected</p> <p>public</p>	<p>Static</p> <p>Final</p> <p>Abstract</p> <p>Synchronized</p> <p>Transient</p> <p>Volatile</p> <p>strictfp</p>

Member Access Modifiers...

- Members can use all four access levels: public, protected, default, private
- Code in one class can access a member of another class depending on the access levels
- If a class cannot be accessed, its members cannot be accessed
- An abstract class cannot be instantiated
- A single abstract method in a class means the whole class must be abstract
- An abstract class can have both abstract and non-abstract methods
- The first concrete class to extend an abstract class must implement all of its abstract methods
- A protected member inherited by a subclass from another package is not accessible to any other class in the subclass package, except for the subclass' own subclasses

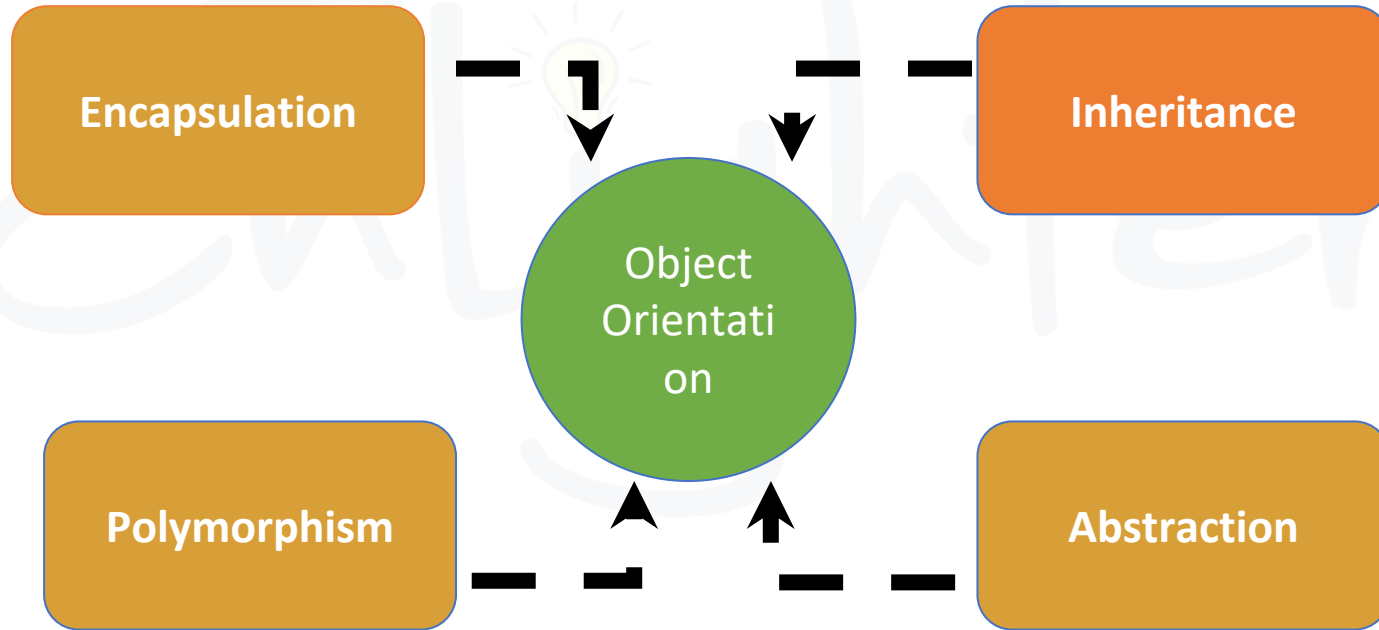
Local variables modifiers...

- Local (method, automatic, or stack) variable declarations cannot have access modifiers
- final is the only modifier available to local variables.
- Local variables don't get default values, so they must be initialized before use.

Variable arguments ...

- As of Java 5, methods can declare a parameter that accepts from zero to many arguments, a so-called var-arg method.
- A var-arg parameter is declared with the syntax `type... name` example:
`doStuff(int... x) { }`
- A var-arg method can have only one var-arg parameter.
- In methods with normal parameters and a var-arg, the var-arg must come last

OOP: Object Oriented Principles



Encapsulation

Concepts...

- Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit
- Encapsulation is also known as Data Hiding
- To achieve encapsulation:
 - Declare the variables of a class as private. Provide public setter and getter methods to modify and view the variables values

Concepts...

Class A

```
B b = new B ();
String nameStr = b.getName ();
b.setName ("new name");
```

```
int serialNum =
b.getSerialNumber ();
b.setSerialNumber(123);
```

Class B

```
getName()
setName()
```

name

```
getSerialNumber()
setSerialNumber()
```

serialNu
m

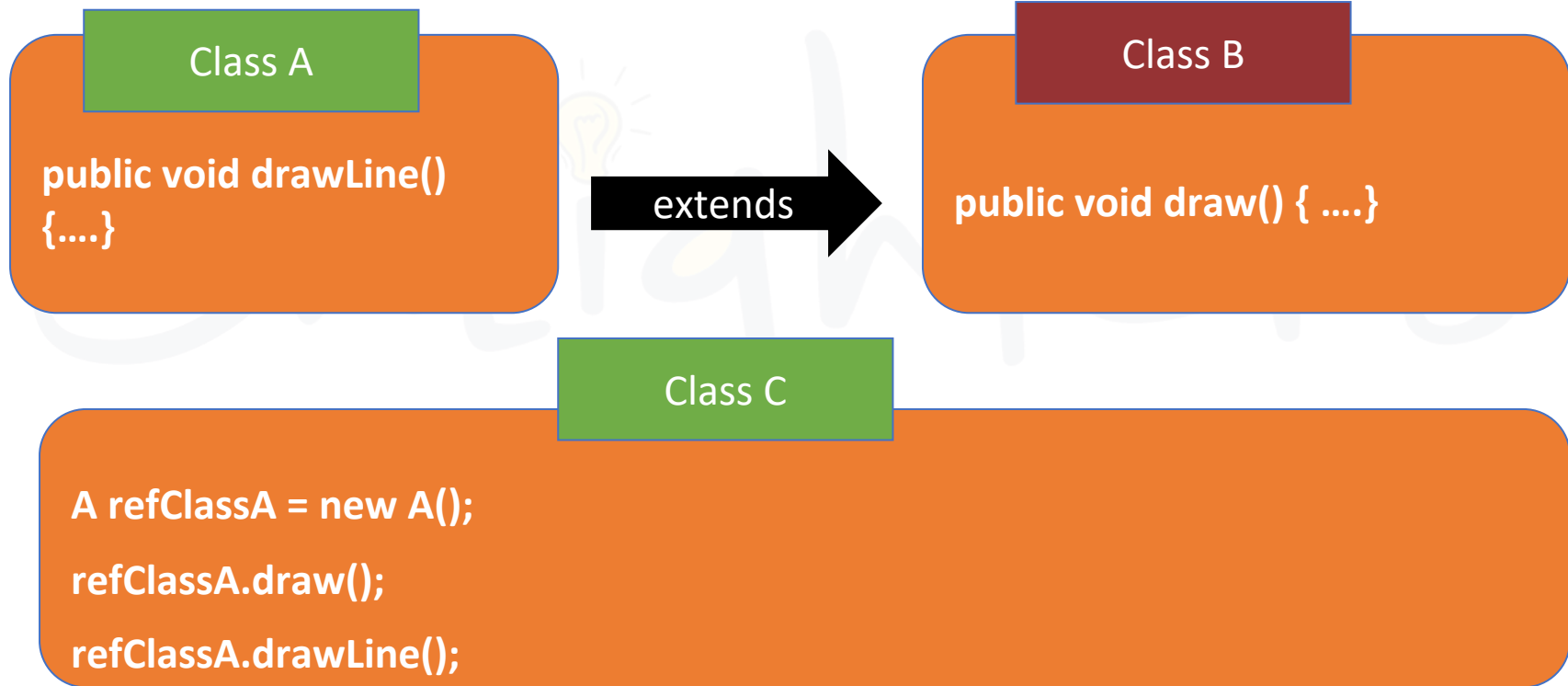
Enlighters

Inheritance

Concepts...

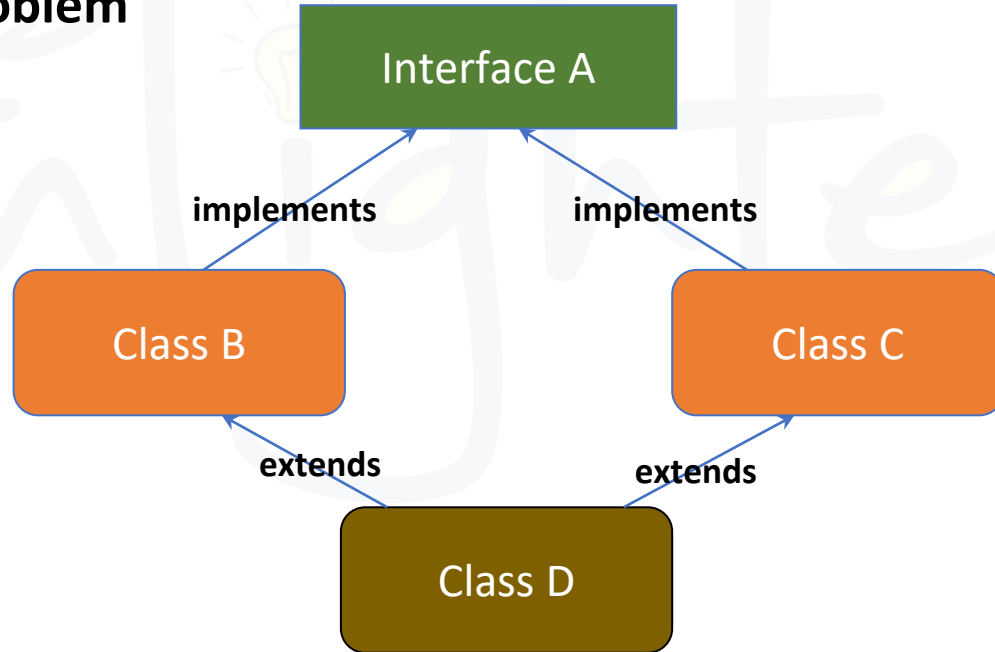
- Inheritance provided mechanism that allowed a class to inherit property of another class
- When class extends another class it inherits all non-private members including fields and methods
- Every class in java inherits from the Object class
- Inheritance Advantage/Disadvantage:
 - Advantage: promotes the code reusability
 - Disadvantage: Two classes (parent and child class) gets tightly coupled
- Multi Inheritance is not supported in Java
- super keyword is used to refer to immediate parent class of a child class

Concepts...



Why multi-inheritance not supported??

Diamond problem



Polymorphism

Concepts...

- Polymorphism refers to the ability to appear in many forms
- Polymorphism in Java is the ability for objects of different classes related by inheritance to response differently to the same function call
- This allows multiple objects of different subclasses – to be treated as objects of a single super class, while automatically selecting the proper methods to apply to a particular object based on the subclass it belongs to
- Example: Car, Audi, BMW

Concepts...

Class Lion

```
public void makeSound() {...}
public int getLegsCount() { return 4; };
```

extends

Class Animal

```
public void makeSound() {...}
public String getType() {.. return "General Animal"; }
```

Class C

```
Animal animalRefObj = new Animal();
Lion lionRefObj = new Lion();
Animal anRefLiObj= new Lion();
animalRefObj .makeSound(); animalRefObj . getType(); animalRefObj .getLegsCount();
lionRefObj .makeSound(); lionRefObj . getType(); lionRefObj .getLegsCount();
anRefLiObj.makeSound(); anRefLiObj. getType(); anRefLiObj.getLegsCount();
```


Reference Variables

- Once declared, a reference variable can be of only one type and that type can never be changed (although the object it references can change)
- A reference variable can refer to any object of the same type as the declared reference or it can refer to any subtype of the declared type
- A reference is a variable, so it can be reassigned to refer other objects (same/subtype), unless the reference is declared final
- Once declared the type of reference variable cannot be changed although object it references can change
- A reference variable's type determines the methods that can be invoked on the object the variable is referencing

Types of Polymorphism

- Runtime Polymorphism (Dynamic Polymorphism)
 - Also known as Dynamic Method Dispatch or **Method Overriding**
 - process in which a call to an overridden method is resolved at runtime rather than at compile time
- Compile time Polymorphism (Static Polymorphism)
 - Also known as **Method Overloading**
 - Method overloading means there are several methods present in a class having the same name but different types/order/number of parameters
- In Java, Polymorphism is achieved by Method Overloading & Method Overriding

NOTE: Method is overridden **not** the data members, so runtime polymorphism can't be achieved by data members.

Runtime Polymorphism

```
class Bank {  
    public void printRateOfInterest() {  
        System.out.println("Rate of interest for bank is 7%");  
    }  
}  
class ICICI extends Bank {  
    public void printRateOfInterest() {  
        System.out.println("Rate of interest for ICICI bank is 10%");  
    }  
}  
class SBI extends Bank {  
    public void printRateOfInterest() {  
        System.out.println("Rate of interest for SBI bank is 9%");  
    }  
}  
public class JavaPolymorphismEx {  
  
    public static void main(String[] args) {  
  
        Bank generalBank = new Bank();  
        generalBank.printRateOfInterest();  
  
        Bank icicBank = new ICICI();  
        icicBank.printRateOfInterest();  
  
        Bank sbiBank = new SBI();  
        sbiBank.printRateOfInterest();  
  
    }  
}
```

```
<terminated> JavaPolymorphismEx [Java Application] /Librar  
Picked up JAVA_TOOL_OPTIONS: -Dfile.enco  
Rate of interest for bank is 7%  
Rate of interest for ICICI bank is 10%  
Rate of interest for SBI bank is 9%
```

Runtime Polymorphism

- **Rules for method overriding**

- The method signature i.e. method name, parameter list and return type have to match exactly
- The overridden method can widen the accessibility but not narrow it, i.e. if it is private in the base class, the child class can make it public but not vice versa
- The return type must be the same as, or a subtype of, the return type declared in the original overridden method in the superclass
- The overriding method CAN throw any unchecked (runtime) exception, regardless of whether the overridden method declares the exception
- The overriding method must NOT throw checked exceptions that are new or broader than those declared by the overridden method
- You cannot override a method marked static or final

Compile time Polymorphism

- **Rules for method overloading**
 - MUST have the same method name
 - MUST change the arguments list
 - Changing the number of arguments OR/AND
 - Changing the type of arguments OR/AND
 - Changing the order of the arguments
 - CAN change the return type or keep it the same
 - CAN change the access modifiers or keep it the same
 - CAN have any variants of exceptions

Compile time Polymorphism

```
public void method1(int arg) {  
  
}  
protected void method1(int arg, float arg2) {  
  
}  
protected void method1(float arg) {  
  
}  
protected void method1(float arg2, int arg) {  
  
}  
}
```

Dynamic & Static Binding

- Connecting a method call to the method body is known as binding.
- There are two types of binding
 - static binding (also known as early binding)
 - When type of the object is determined at compiled time (by the compiler), it is known as static binding.
 - If there is any private, final or static method in a class, there is static binding
 - dynamic binding (also known as late binding)
 - When type of the object is determined at run-time, it is known as dynamic binding

Enlighters

Abstraction

Concepts...

- Abstraction is a process of hiding the implementation details from the user and instead only the functionality will be provided
- In other words, the user will have the information on what the object does instead of how it does it
- In Java, abstraction is achieved using Abstract classes and interfaces
- Encapsulation is data hiding(information hiding) while Abstraction is detail hiding(implementation hiding)

Abstraction rules

- Abstract classes may or may not contain abstract methods, i.e., methods without body (`public void get();`)
- But, if a class has at least one abstract method, then the class must be declared abstract
- If a class is declared abstract, it cannot be instantiated
- To use an abstract class, you have to inherit it from another class, provide implementations to the abstract methods in it
- If you inherit an abstract class, you have to provide implementations to all the abstract methods in it