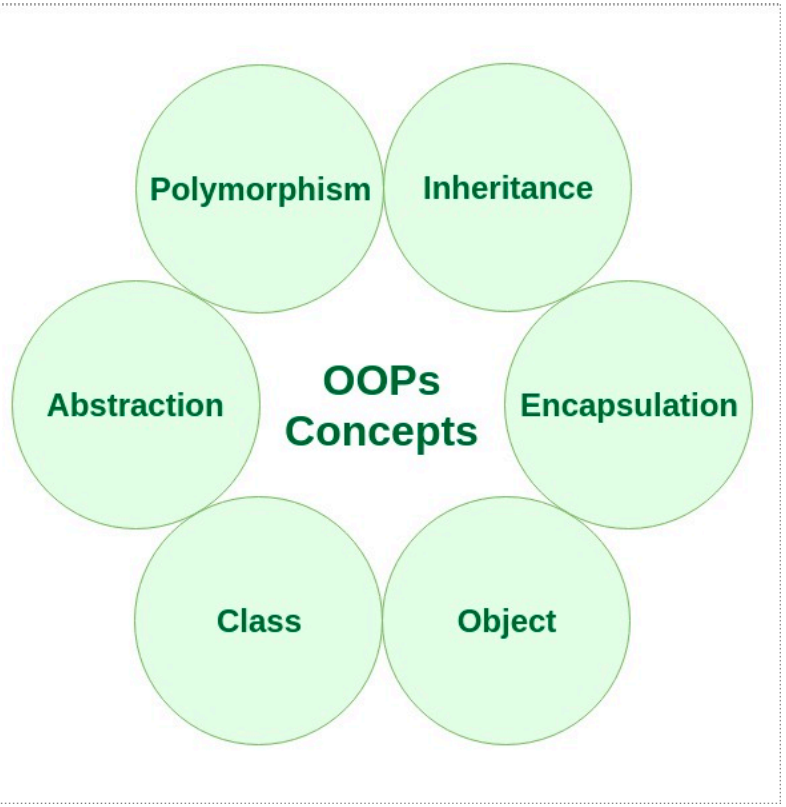
Object Oriented Programming (OOPs)

What to Expect in This Module



Class

Object

Method

Polymorphism

Inheritance

Encapsulation

Abstraction

Classes and Objects in Java

- **Modifiers** : A class can be public or has default access
- **Class name:** The name should begin with a initial letter (capitalized by convention)
- **Superclass(if any):** The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent
- **Interfaces(if any):** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
- **Body:** The class body surrounded by braces, { }.

Access Modifiers

Java Access Levels

Modifier	Class	Package	Subclass	World
`public`	Y	Y	Y	Y
`protected`	Y	Y	**Y**	N
no modifier	Y	Y	**N**	N
`private`	Y	N	N	N

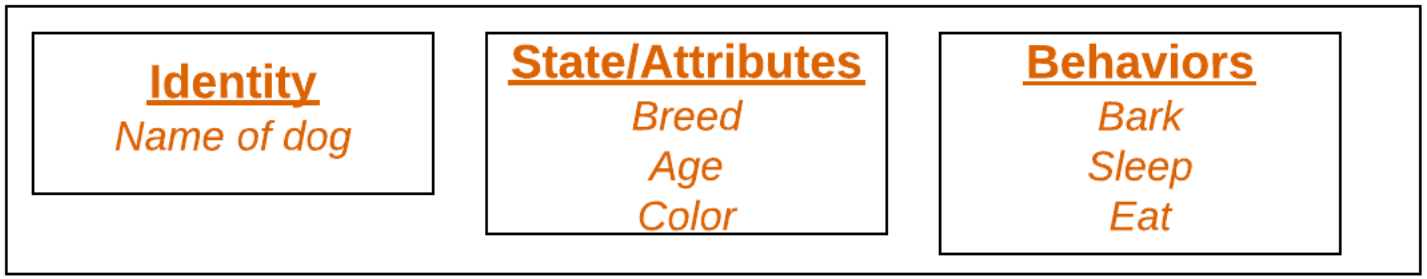
Biggest difference between 'protected' and 'nothing' is the subclass access.

Rules to Create a Class

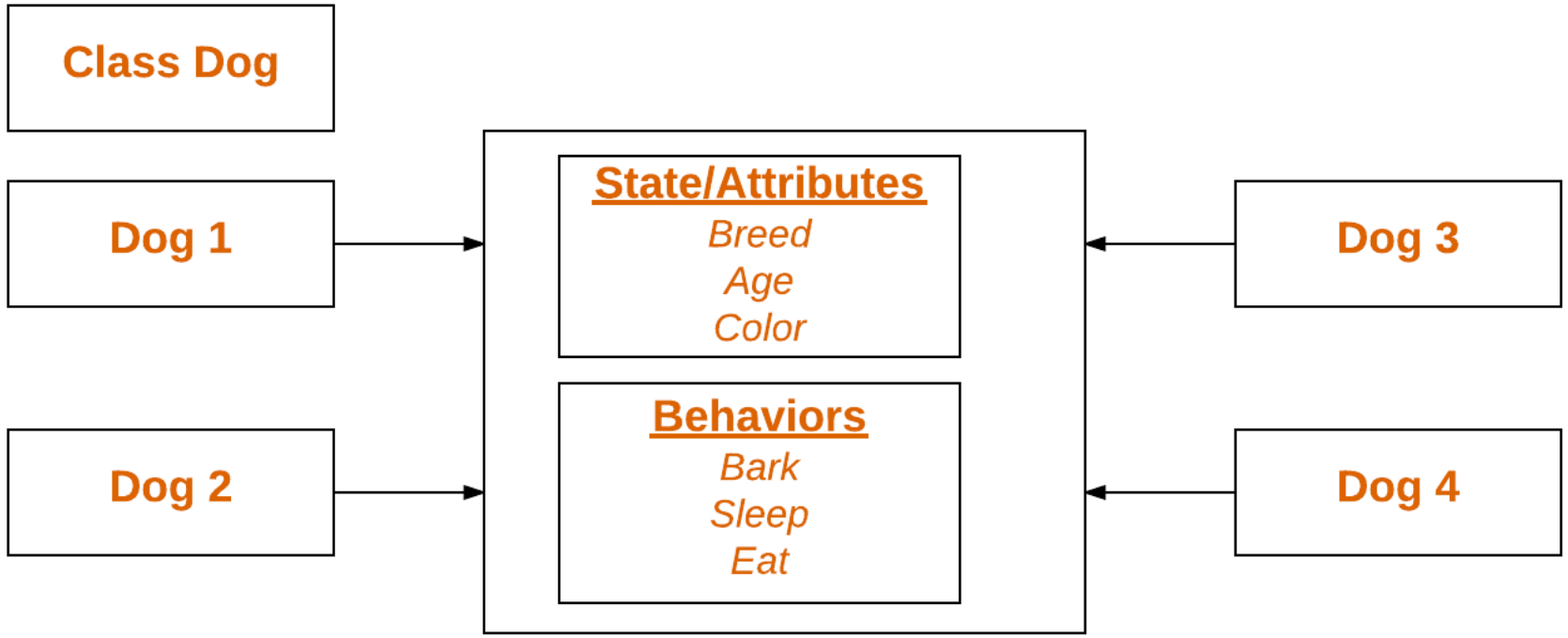
1. A Java class must have the class keyword followed by the class name, and class must be followed by a legal identifier.
2. The class name must start with a capital letter and if you are using more than one word to define a class name, every first letter of the latter words should be made capital.
3. There should not be any spaces or special characters used in a class name except the dollar symbol($) and underscore(_).
4. A Java class can only have public or default access specifier.
5. It must have the class keyword, and class must be followed by a legal identifier.
6. It can extend only one parent class. By default, all the classes extend java.lang.Object directly or indirectly.
7. A class may optionally implement any number of interfaces separated by commas.
8. The class's members must be always declared within a set of curly braces {}.
9. Each ***.java*** source file can contain any number of default classes but can only have one public class.
10. Class containing the main() method is known as the Main class as it will act as the entry point in program.

Objects

- **State** : It is represented by attributes of an object. It also reflects the properties of an object.
- **Behavior** : It is represented by methods of an object. It also reflects the response of an object with other objects.
- **Identity** : It gi ɔ interact with other objects.



Declaring Objects
(Also called instantiating a class)



Ways to create object of a class

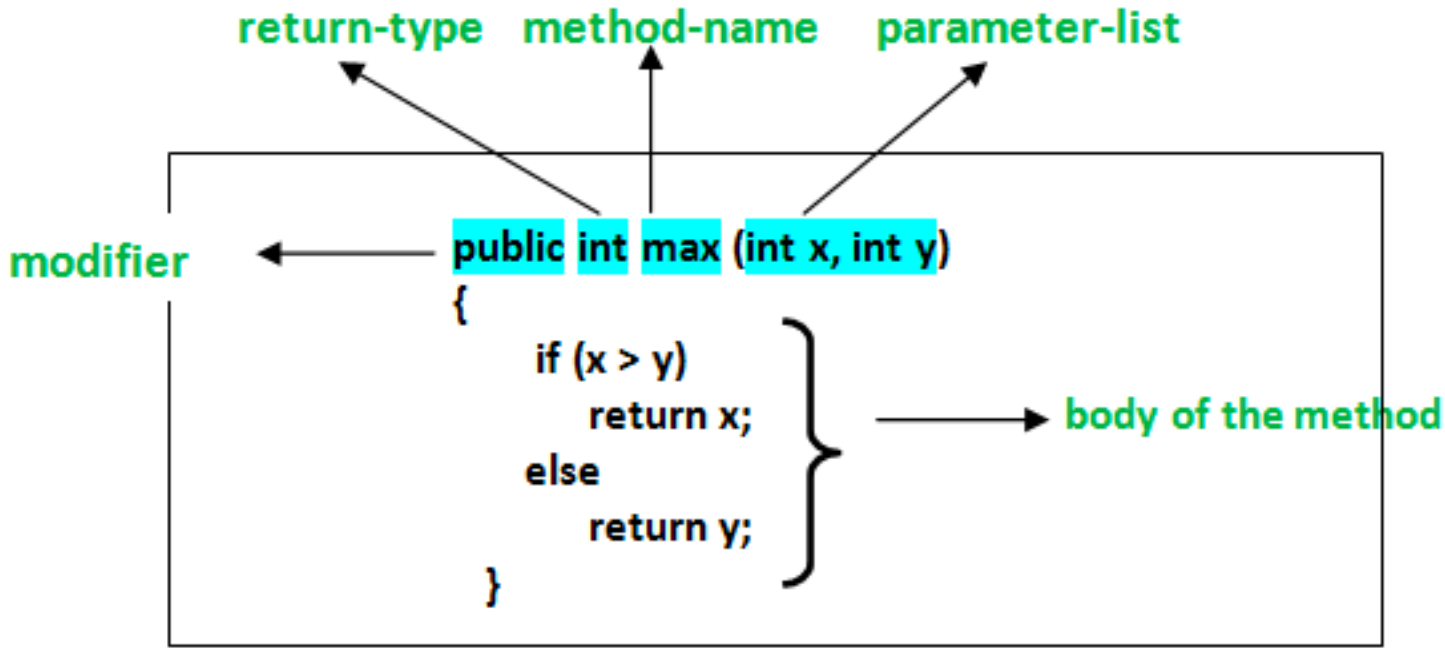
1. **Using new keyword**
2. **Using Class.forName(String className) method**
3. **Using clone() method**

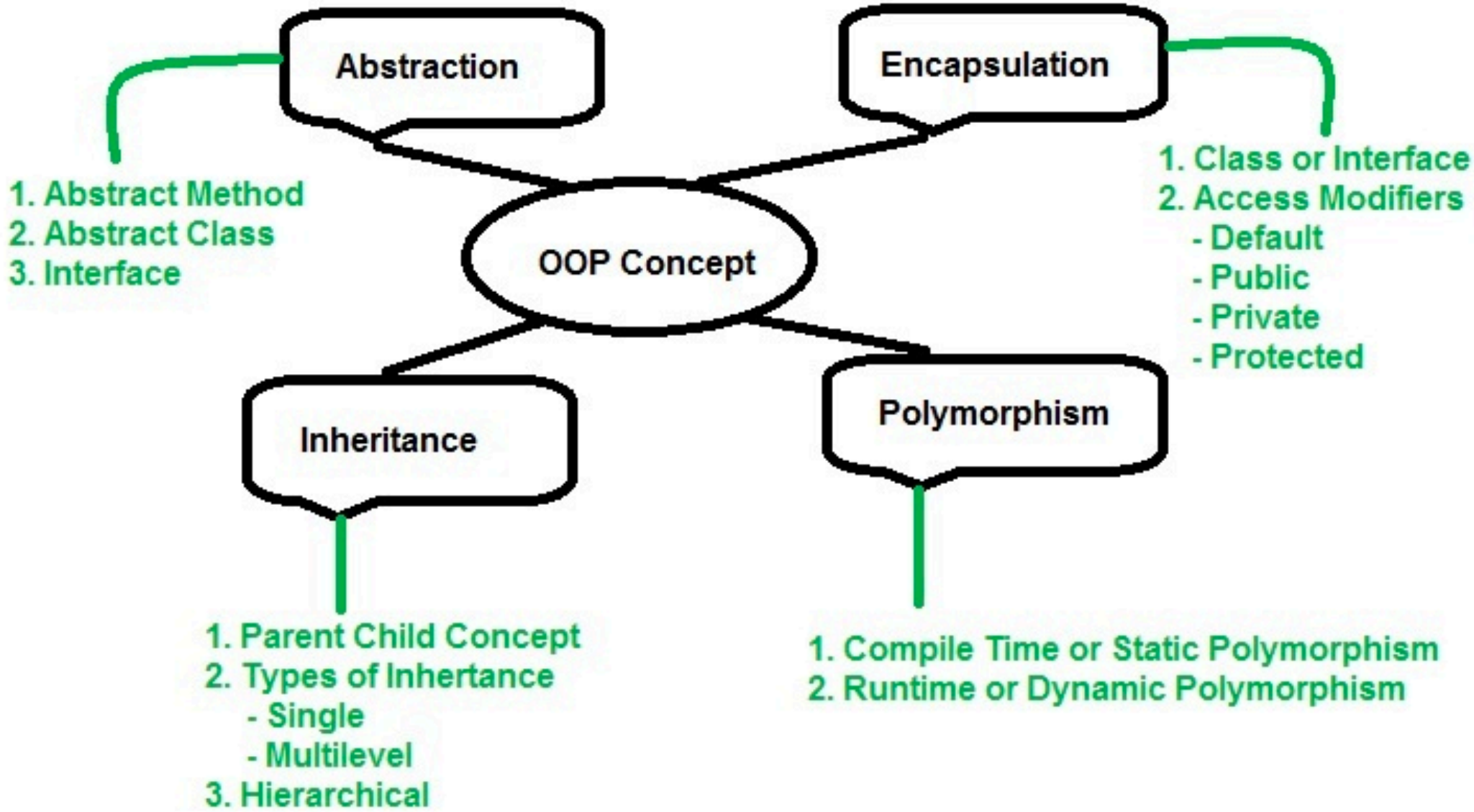
Constructors in Java

1. **What is default constructor ?**
2. **When is a Constructor called ?**
3. **What rules for writing Constructor?**
   - *Constructor(s) of a class must has same name as the class name in which it resides.*
   - *A constructor in Java can not be abstract, final, static and Synchronized.*
   - *Access modifiers can be used in constructor declaration to control its access i.e which other class can call the constructor.*
4. **What types Constructors ?**
   - *No-argument constructor*
   - *Parameterized Constructor*
5. **Does constructor return any value?**
6. **How constructors are different from methods in Java?**
   - *Constructor(s) must have the same name as the class within which it defined while it is not necessary for the method in java.*
   - *Constructor(s) do not return any type while method(s) have the return type or void if does not return any value.*
   - *Constructor is called only once at the time of Object creation while method(s) can be called any numbers of time.*

Methods in Java

1. **Modifier**
2. **The return type**
3. **Method Name**
4. **Parameter list**
5. **Exception list**
6. **Method body**




Abstraction
1. Abstract Method
2. Abstract Class
3. Interface
Encapsulation
1. Class or Interface
2. Access Modifiers
- Default
- Public
- Private
- Protected
OOP Concept
Inheritance
1. Parent Child Concept
2. Types of Inhertance
- Single
- Multilevel
3. Hierarchical
Polymorphism
1. Compile Time or Static Polymorphism
2. Runtime or Dynamic Polymorphism

Polymorphism

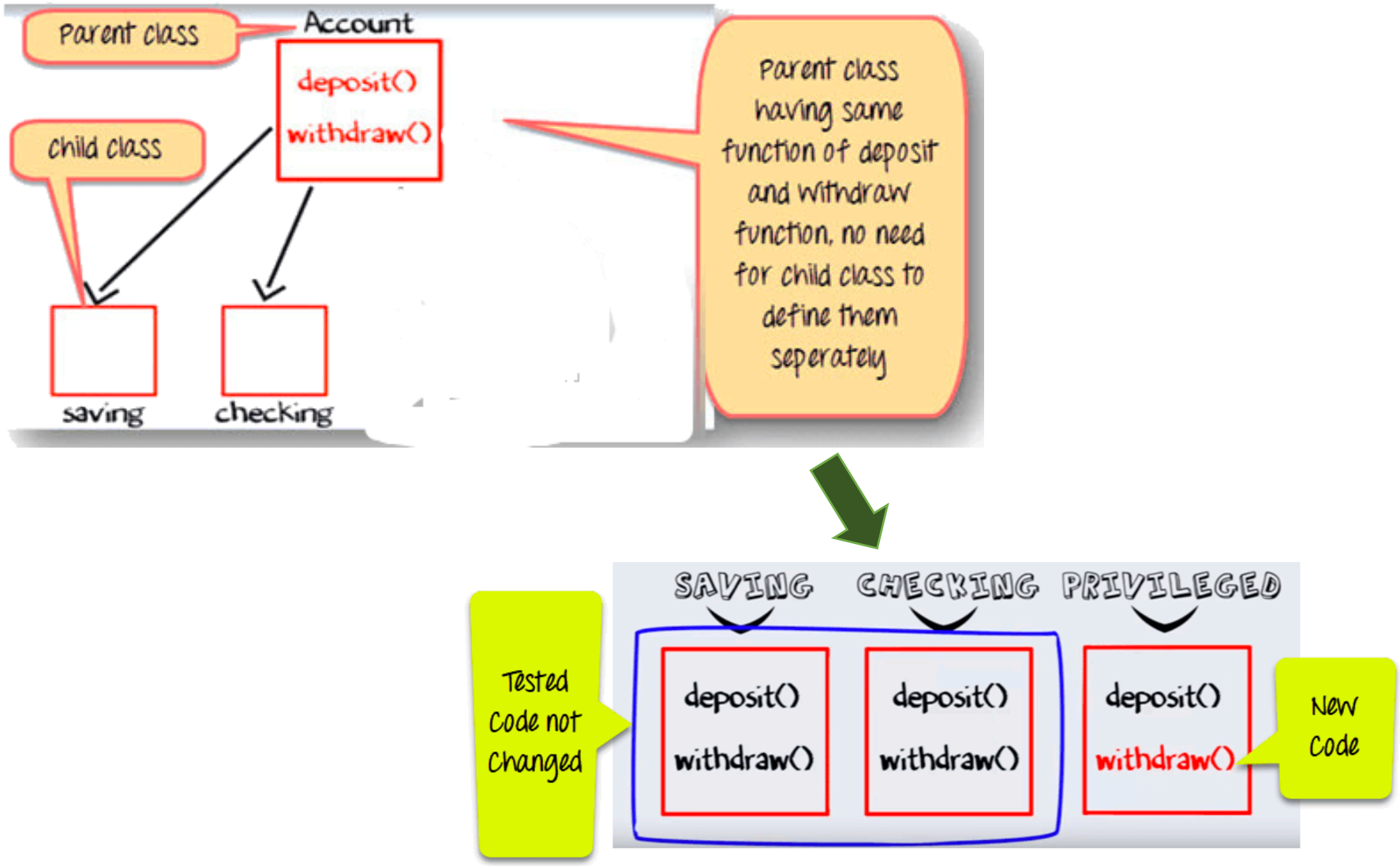
1. What is Polymorphism?
2. Java Polymorphism in OOP’s with Example
3. Method Overriding
4. Difference between Overloading and Overriding
5. What is Dynamic Polymorphism?
6. Super Keyword
7. Difference between Static & Dynamic Polymorphism

What is Polymorphism?

- one name can have many forms

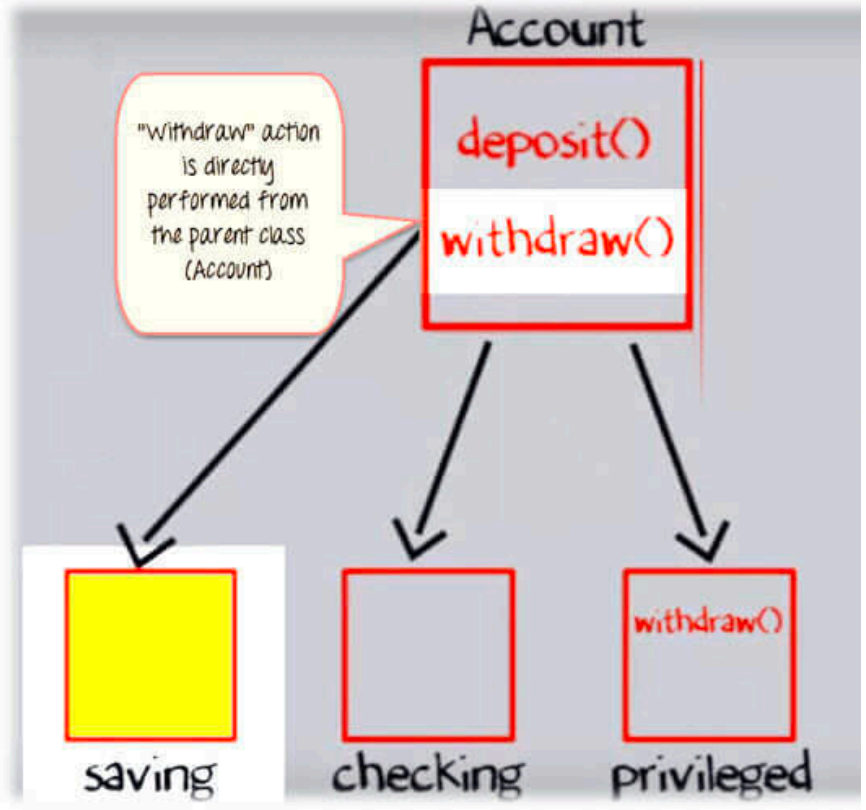
For example, you have a smartphone for communication. The communication mode you choose could be anything. It can be a call, a text message, a picture message, mail, etc. So, the goal is common that is communication, but their approach is different.

Example

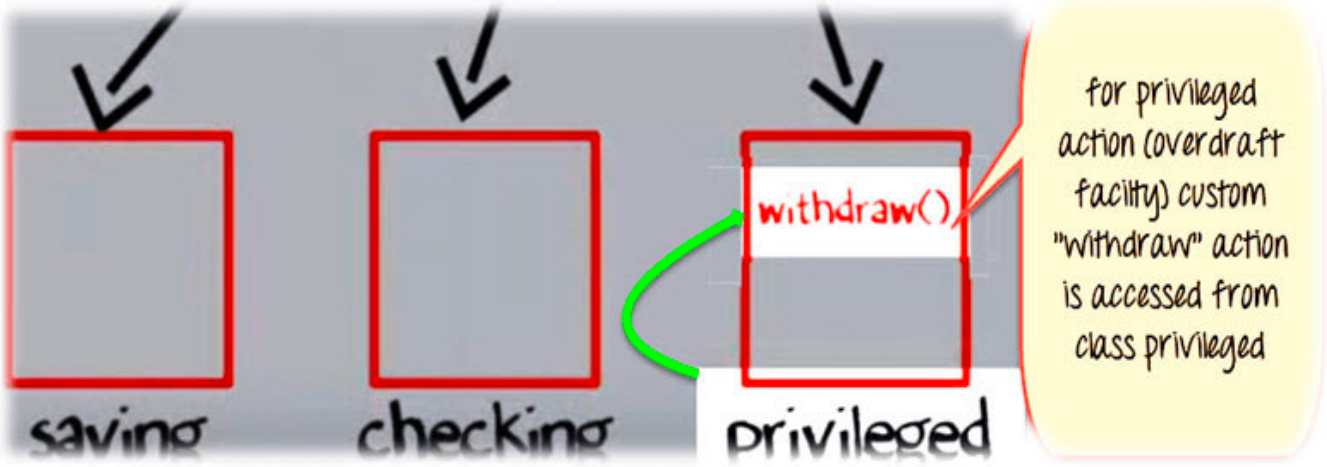

Parent class
Account
deposit()
withdraw()
child class
saving
checking
Parent class having same function of deposit and withdraw function, no need for child class to define them seperately
SAVING
CHECKING
PRIVILEGED
Tested Code not Changed
deposit()
withdraw()
deposit()
withdraw()
deposit()
withdraw()
New Code

Example…

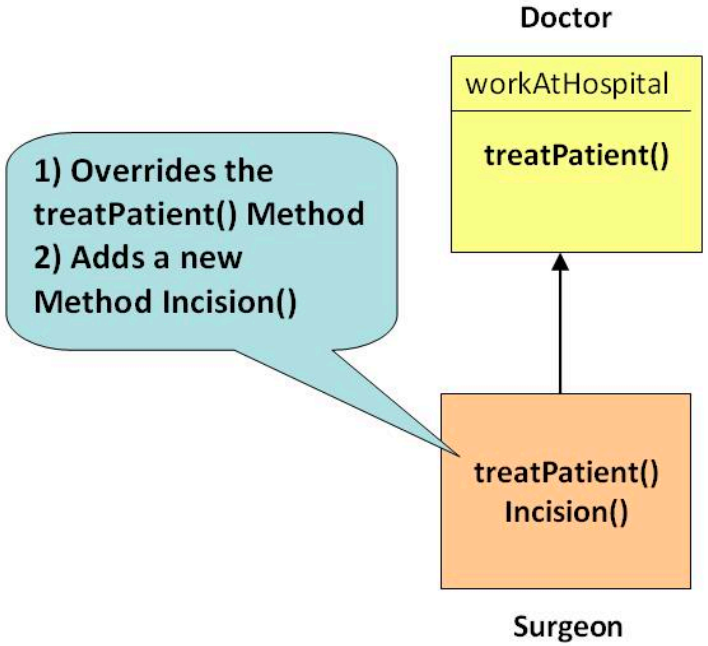
When the "withdrawn" method for saving account is called a method from parent account class is executed



When the "Withdraw" method for the privileged account (overdraft facility) is called withdraw method defined in the privileged class is executed



Method Overriding



Rules for Method Overriding

1. The method signature i.e. method name, parameter list and return type have to match exactly.
2. 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.

Difference between Overloading and Overriding

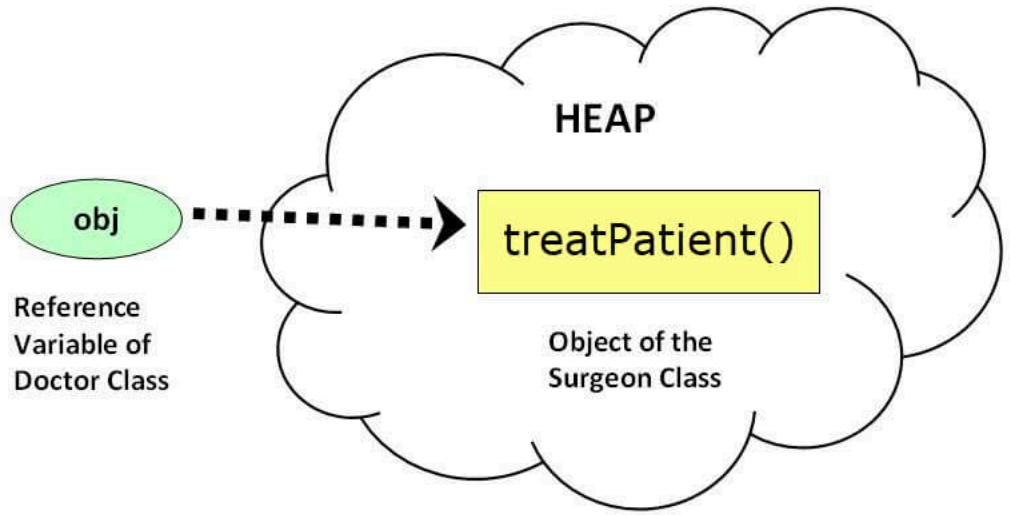
Method Overloading	Method Overriding
Method overloading is in the same class, where more than one method have the same name but different signatures.	Method overriding is when one of the methods in the super class is redefined in the sub-class. In this case, the signature of the method remains the same.

Ex:
`void sum (int a , int b);`
`void sum (int a , int b, int c);`
`void sum (float a, double b);`
	Ex:
	`class X{`
	`  public int sum(){`
	`  // some code`
	`  }`
	`}`
	
	`class Y extends X{`
	`  public int sum(){`
	`  //overridden method`
	`  //signature is same`
	`  }`
	`}`

What is Dynamic Polymorphism?

The mechanism by which multiple methods can be defined with same name and signature in the superclass and subclass. The call to an overridden method are resolved at run time

```
Doctor obj = new Surgeon();

obj.treatPatient();
```



Super Keyword

```
treatPatient(){
   super.treatPatient();
     //add code specific to Surgeon
}
```

Static vs Dynamic Polymorphism

Static Polymorphism	Dynamic Polymorphism
It relates to method overloading.	It relates to method overriding.

Errors, if any, are resolved at compile time. Since the code is not executed during compilation, hence the name static.
 	
Ex:
 	
`void sum (int a , int b);`
`void sum (float a, double b);`
`int sum (int a, int b); //compiler gives error.`
	In case a reference variable is calling an overridden method, the method to be invoked is determined by the object, your reference variable is pointing to. This is can be only determined at runtime when code in under execution, hence the name dynamic.
	
	Ex:
	
	`//reference of parent pointing to child object`
	`Doctor obj = new Surgeon();`
	`// method of child called`
	`obj.treatPatient();`

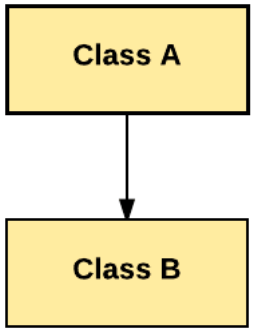
What is Inheritance?

- Mechanism in which one class acquires the property of another class.
- Facilitates Reusability

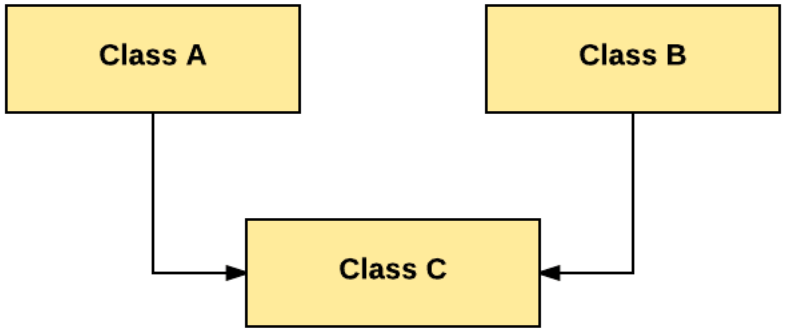
- For example, a child inherits the traits of his/her parents. With inheritance, we can reuse the fields and methods of the existing class.

Types of Inheritance

Single Inheritance.

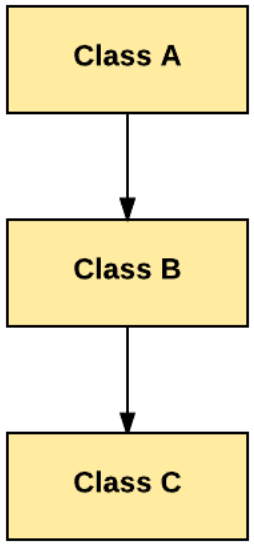


Multiple Inheritance

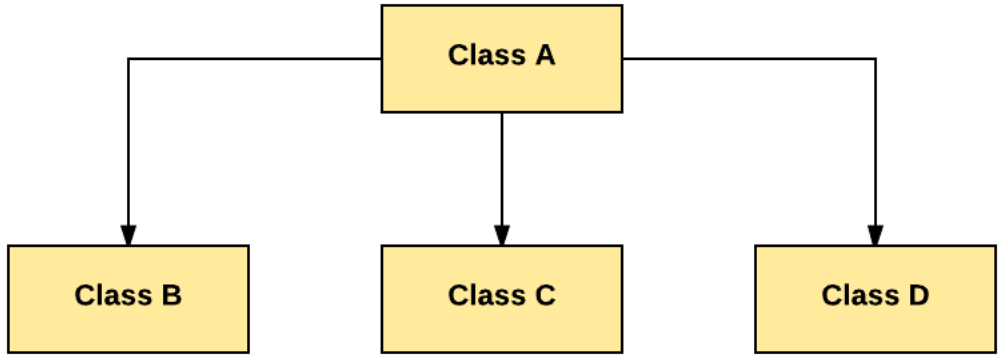


Types of Inheritance

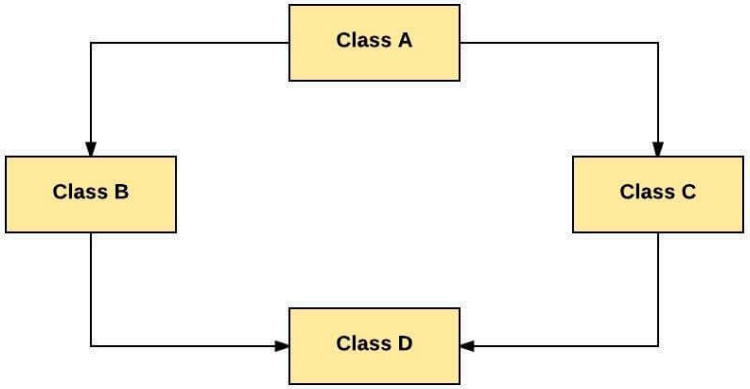
Multilevel Inheritance



Hierarchical Inheritance

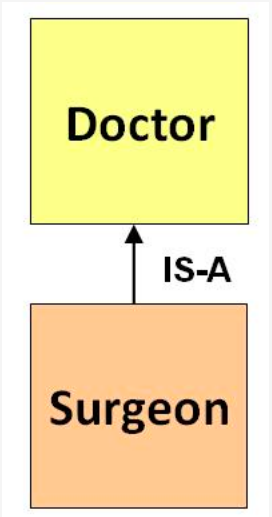


Hybrid Inheritance



Examples

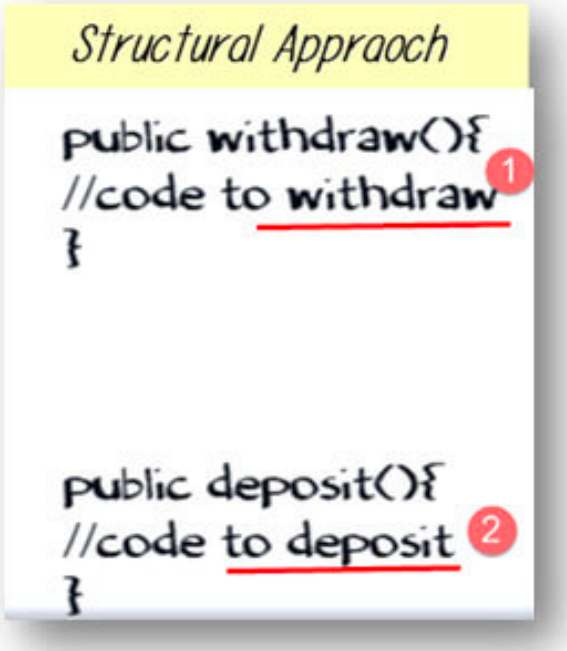
```
class Doctor {
 void Doctor_Details() {
  System.out.println("Doctor Details...");
 }
}

class Surgeon extends Doctor {
 void Surgeon_Details() {
  System.out.println("Surgen Detail...");
 }
}

public class Hospital {
 public static void main(String args[]) {
  Surgeon s = new Surgeon();
  s.Doctor_Details();
  s.Surgeon_Details();
 }
}
```



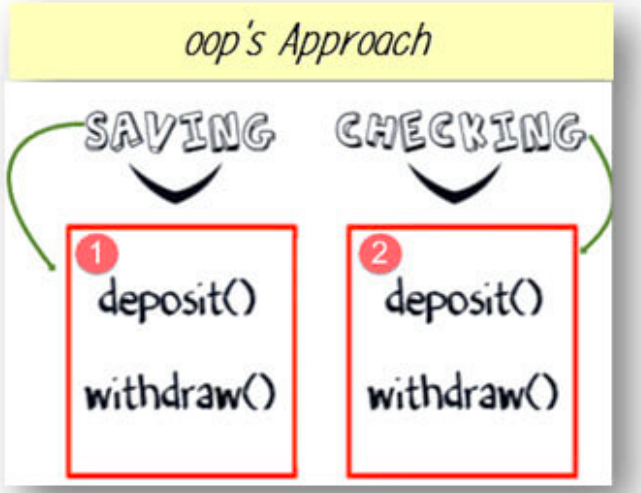
Previous Example

1) saving
2) Current / Checking

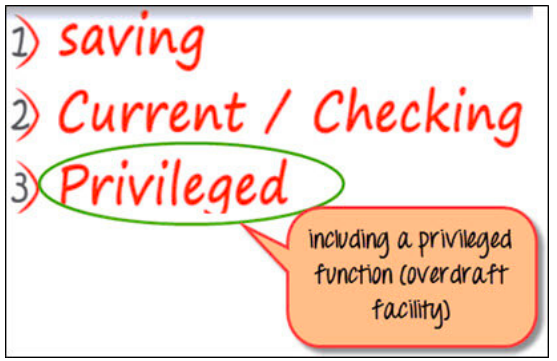
Structural approach



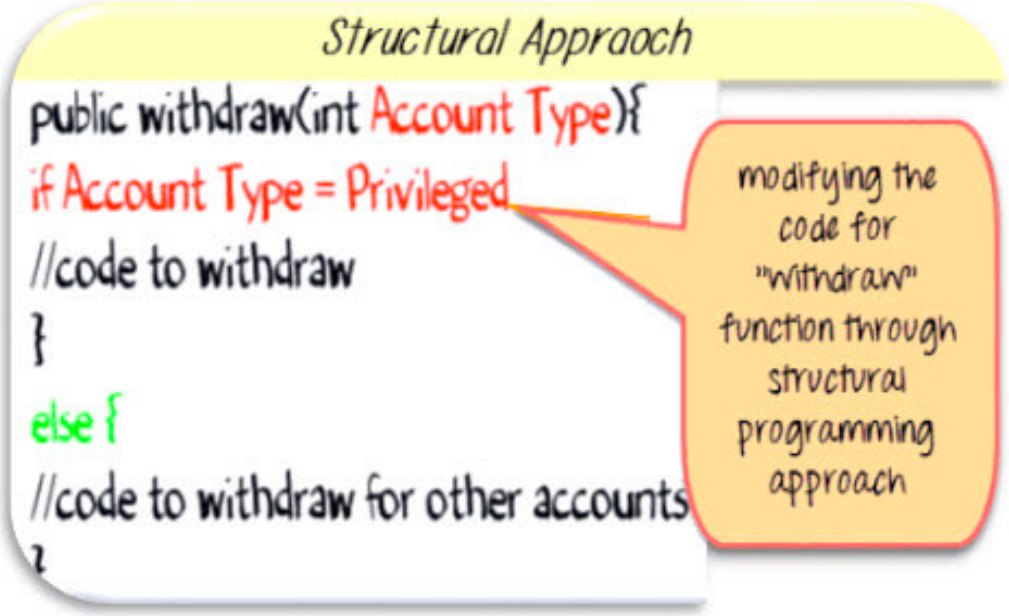
OOP's approach



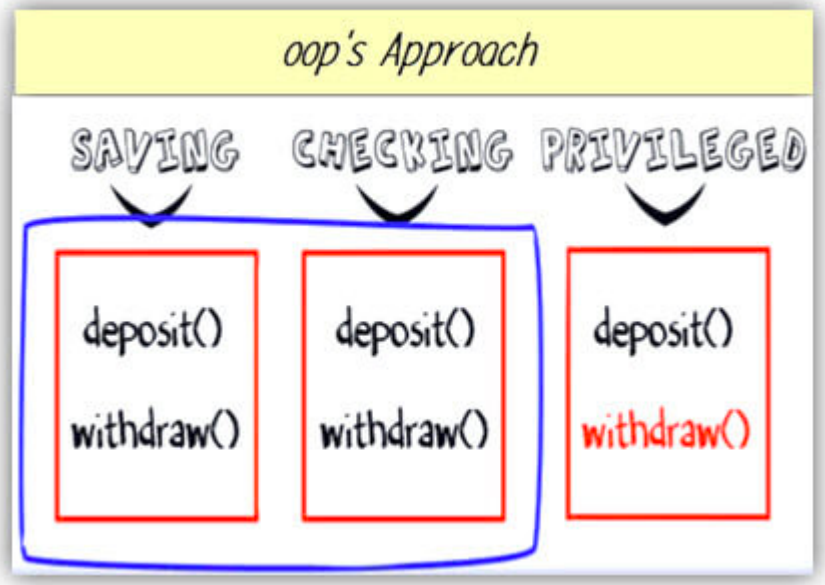
Change Request in Software



Structural approach



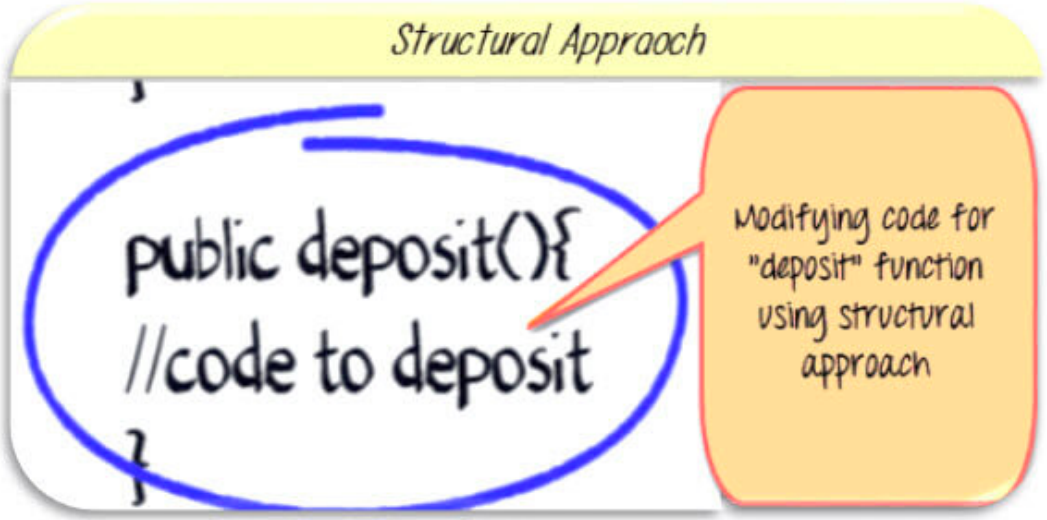
OOP's approach



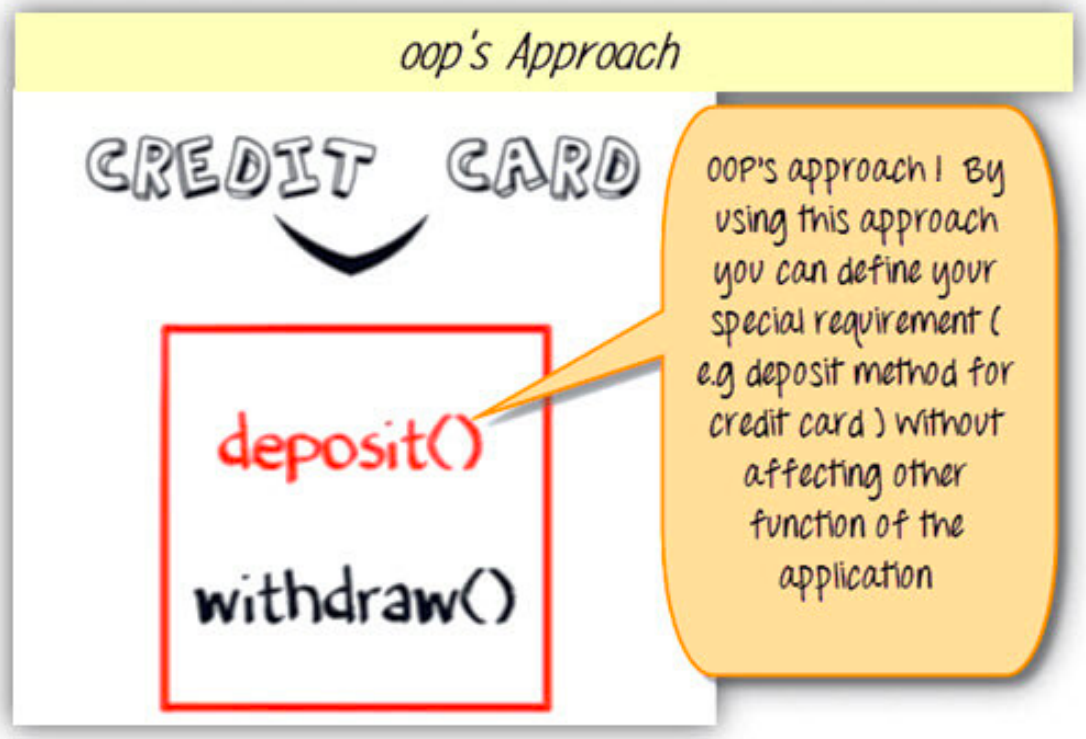
Another Change Request

1) saving
2) Current / Checking
3) Privileged
4) Credit Card

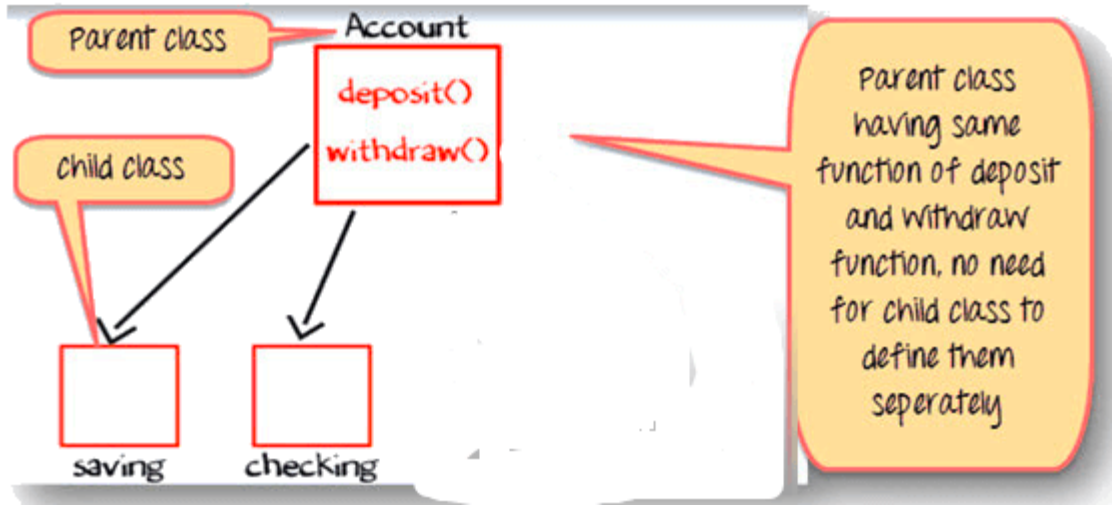
Structural approach

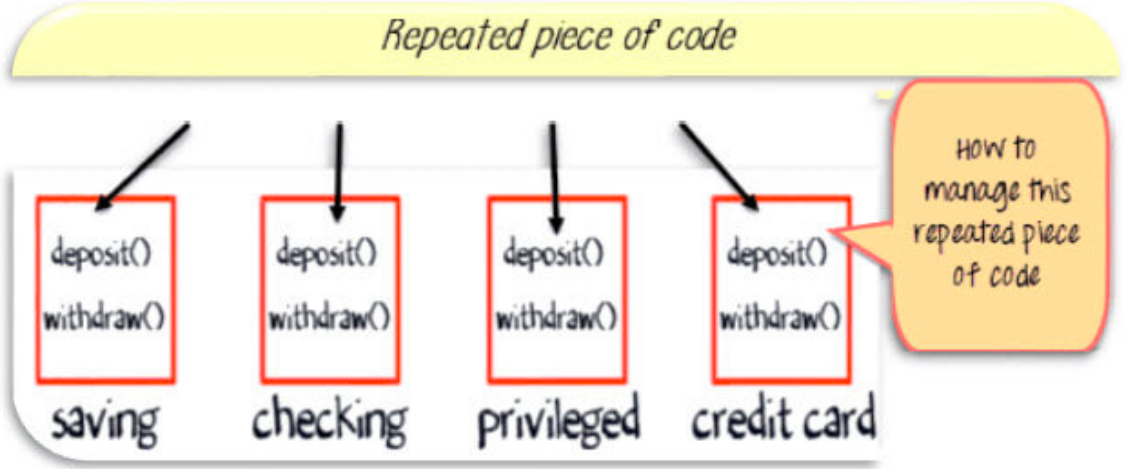


OOP's approach



Advantage of Inheritance



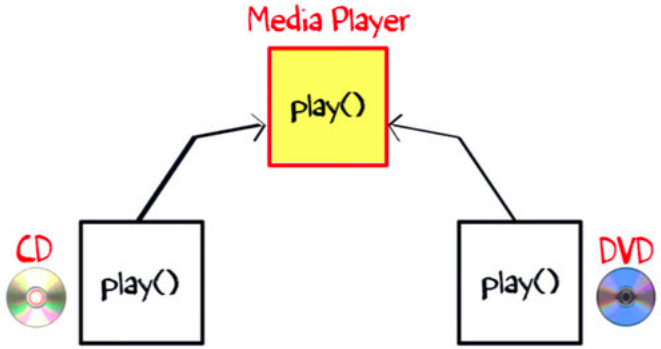


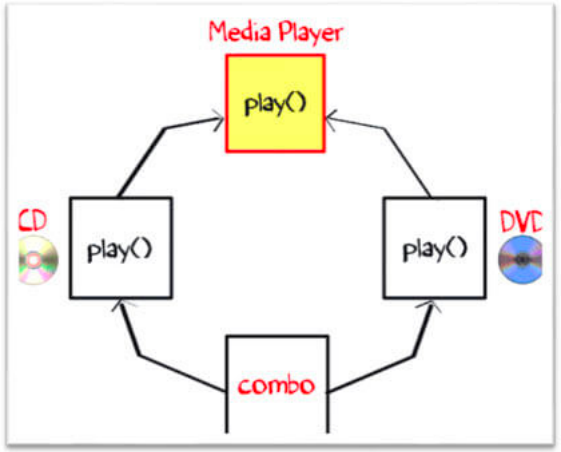
What is Interface ?

- Like Java Class
- But it only has static constants and abstract method.
- Java uses Interface to implement multiple inheritance.
- A Java class can implement multiple Java Interfaces.
- All methods in an interface are implicitly public and abstract.

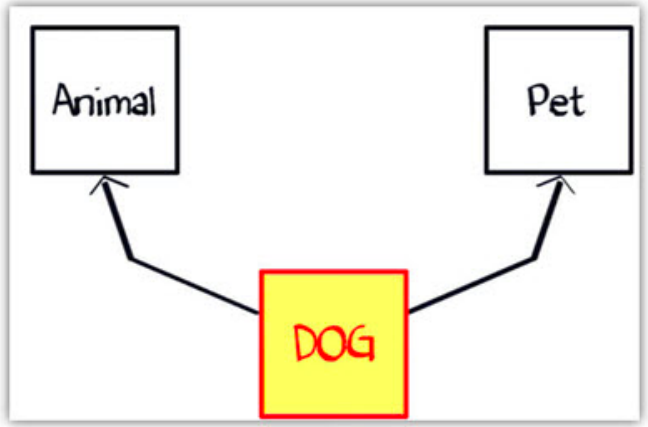
Why is an Interface required?

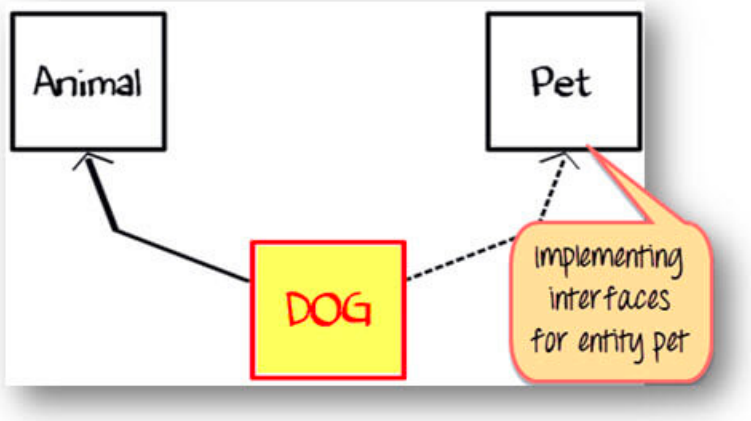
Example 1





Example 2





Class vs Interfaces

Class	Interface
In class, you can instantiate variable and create an object.	In an interface, you can't instantiate variable and create an object.
Class can contain concrete(with implementation) methods	The interface cannot contain concrete(with implementation) methods
The access specifiers used with classes are private, protected and public.	In Interface only one specifier is used- Public.

Must know facts about Interface

- A Java class can implement multiple Java Interfaces. It is necessary that the class must implement all the methods declared in the interfaces.
- Class should override all the abstract methods declared in the interface
- The interface allows sending a message to an object without concerning which classes it belongs.
- Class needs to provide functionality for the methods declared in the interface.
- All methods in an interface are implicitly public and abstract
- An interface cannot be instantiated
- An interface reference can point to objects of its implementing classes
- An interface can extend from one or many interfaces. Class can extend only one class but implement any number of interfaces
- An interface cannot implement another Interface. It has to extend another interface if needed.
- An interface which is declared inside another interface is referred as nested interface
- At the time of declaration, interface variable must be initialized. Otherwise, the compiler will throw an error.
- The class cannot implement two interfaces in java that have methods with same name but different return type.

New Features added in Java 8 & 9

Java 8

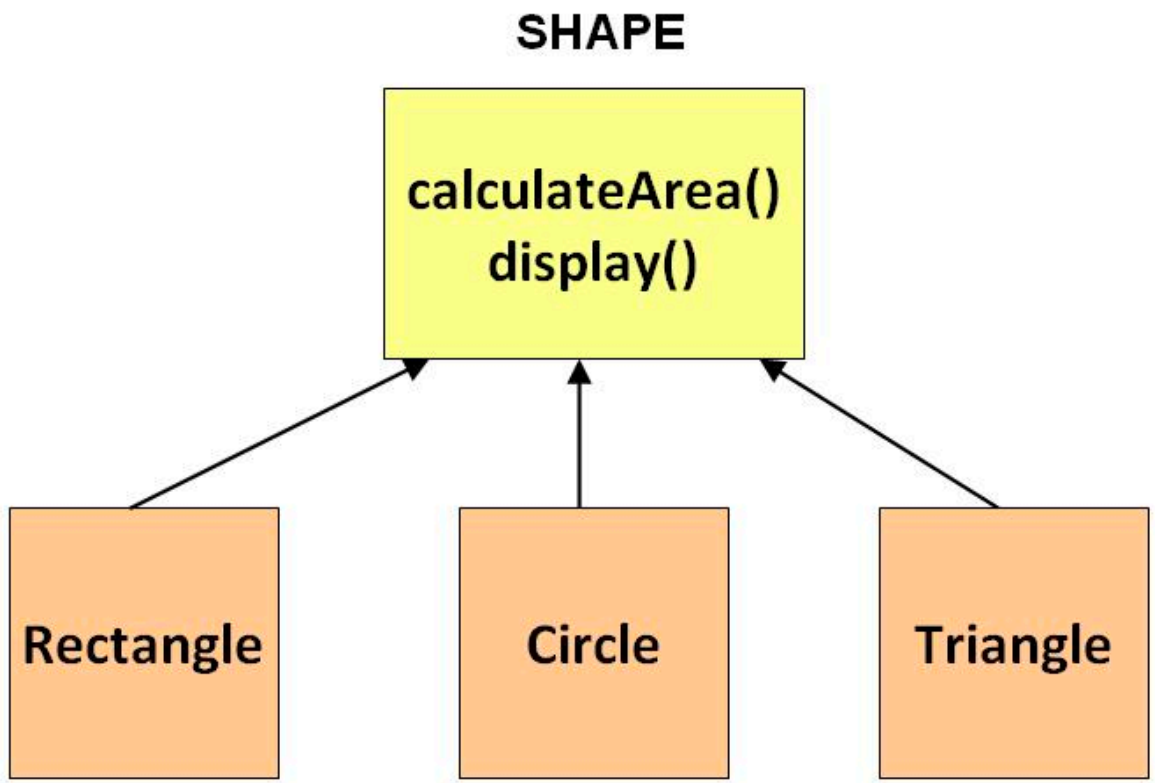
- ✓ Default implementation
- ✓ Define static methods

Java 9 – Remind Me

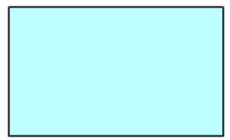
- ✓ Constant variables
- ✓ Abstract methods
- ✓ Default methods
- ✓ Static methods
- ✓ Private methods
- ✓ Private Static methods

What is Abstract Class?

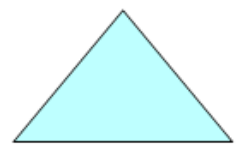
Abstract Classes are classes in Java, that declare one or more abstract methods



Rectangle obj = new Rectangle();



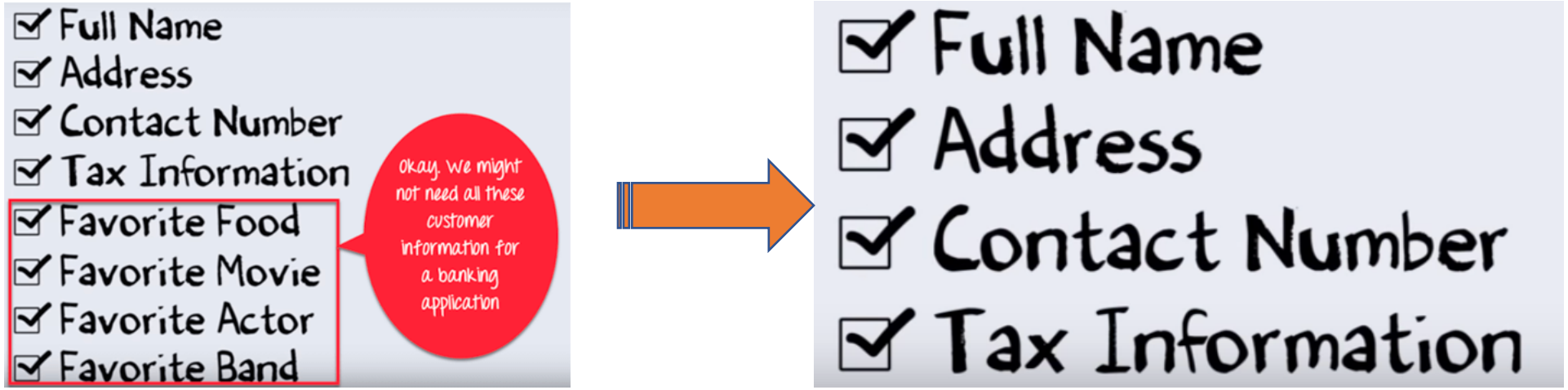
Triangle obj = new Triangle();



Shape obj = new Shape();

???

Key Concept


Full Name
Address
Contact Number
Tax Information
Favorite Food
Favorite Movie
Favorite Actor
Favorite Band
Okay. We might not need all these customer information for a banking application
Full Name
Address
Contact Number
Tax Information

What are Abstract Methods?

Is a method that has just the method definition but does not contain implementation

Important Points

- An abstract class **may** also have concrete (complete) methods.
- For design purpose, a class can be declared abstract even if it does not contain any abstract methods
- Reference of an abstract class can point to objects of its sub-classes thereby achieving run-time polymorphism Ex: Shape obj = new Rectangle();
- A class must be compulsorily labeled abstract, if it has one or more abstract methods.

Important Reasons For Using Interfaces

- ✓ Interfaces are used to achieve abstraction.
- ✓ Designed to support dynamic method resolution at run time
- ✓ It helps you to achieve loose coupling.
- ✓ Allows you to separate the definition of a method from the inheritance hierarchy

Important Reasons For Using Abstract Class

- ✓ Abstract classes offer default functionality for the subclasses.
- ✓ Provides a template for future specific classes
- ✓ Helps you to define a common interface for its subclasses
- ✓ Abstract class allows code reusability.

Final Keyword

- A final class can not be inherited
- A final variable becomes a constant and its value can not be changed
- A final method can not be overridden. This is done for security reasons, and these methods are used for optimization

Abstraction Vs. Encapsulation

Abstraction	Encapsulation
Abstraction solves the issues at the design level.	Encapsulation solves it implementation level.
Abstraction is about hiding unwanted details while showing most essential information.	Encapsulation means binding the code and data into a single unit.
Abstraction allows focussing on what the information object must contain	Encapsulation means hiding the internal details or mechanics of how an object does something for security reasons.

Interface Vs. Abstract Class

Parameters	Interface	Abstract class
Speed	Slow	Fast
Multiple Inheritances	Implement several Interfaces	Only one abstract class
Structure	Abstract methods	Abstract & concrete methods
When to use	Future enhancement	To avoid independence
Inheritance/ Implementation	A Class can implement multiple interfaces	The class can inherit only one Abstract Class
Default Implementation	While adding new stuff to the interface, it is a nightmare to find all the implementors and implement newly defined stuff.	In case of Abstract Class, you can take advantage of the default implementation.
Access Modifiers	The interface does not have access modifiers. Everything defined inside the interface is assumed public modifier.	Abstract Class can have an access modifier.
When to use	It is better to use interface when various implementations share only method signature. Polymorphic hierarchy of value types.	It should be used when various implementations of the same kind share a common behavior.
Data fields	the interface cannot contain data fields.	the class can have data fields.

Interface Vs. Abstract Class Cont...

Implementation	An interface is abstract so that it can't provide any code.	An abstract class can give complete, default code which should be overridden.
Use of Access modifiers	You cannot use access modifiers for the method, properties, etc.	You can use an abstract class which contains access modifiers.
Usage	Interfaces help to define the peripheral abilities of a class.	An abstract class defines the identity of a class.
Defined fields	No fields can be defined	An abstract class allows you to define both fields and constants
Inheritance	An interface can inherit multiple interfaces but cannot inherit a class.	An abstract class can inherit a class and multiple interfaces.
Constructor or destructors	An interface cannot declare constructors or destructors.	An abstract class can declare constructors and destructors.
Limit of Extensions	It can extend any number of interfaces.	It can extend only one class or one abstract class at a time.
Abstract keyword	In an abstract interface keyword, is optional for declaring a method as an abstract.	In an abstract class, the abstract keyword is compulsory for declaring a method as an abstract.
Class type	An interface can have only public abstract methods.	An abstract class has protected and public abstract methods.

1. When to use abstraction ?

when you know something needs to be there but not sure how exactly it should look like

2. Why abstraction is important?

- ✓ Suppose that abstraction is not there, we cannot identify an object via its class name.
- ✓ If abstraction is not there, we cannot access the attributes of an object.
- ✓ If abstraction is not there, we cannot invoke the behaviors of an object.
- ✓ Abstraction is the most fundamental concept on which others rely on, such as encapsulation, inheritance and polymorphism.

What is Encapsulation in Java?

Encapsulation is a principle of wrapping data (variables) and code together as a single unit

Approach 1

```
class Hacker{
Account a= new Account ();
a.account_balance= -100;
```

```
class Account{
private int account_number;
private int account_balance;

public void show Data(){
// code to show data
}

public void deposit(int a){
```

```
class Hacker{
Account a= new Account ();
a.account_balance= -100;

}
```

Approach 2

```
class Hacker{
Account a= new Account
a.account_balance= -100
a.deposit(-100);
}
```

```
public void deposit(int a){
if (a<0){
//show error
}
else
account_balance=
}
}
```

method implementation has checked for negative values (-100) and throws an

```
class Account{
private int account_number;
private int account_balance;

public void show Data(){
// code to show data
}

public void deposit(int a){
if (a<0){
//show error
}
else
account_balance= account_balance+a;
}
}
```

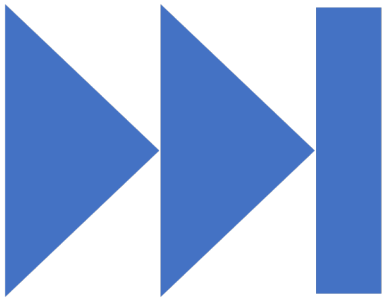
The entire code can be thought of a capsule, and can be only communicate through messages

Abstraction vs. Encapsulation

- ✓ Encapsulation is more about "How" to achieve a functionality
- ✓ Abstraction is more about "What" a class can do.

Advantages

- ✓ Encapsulation is binding the data with its related functionalities. Here functionalities mean "methods" and data means "variables"
- ✓ So we keep variable and methods in one place. That place is "class." Class is the base for encapsulation.
- ✓ With Java Encapsulation, you can hide (restrict access) to critical data members in your code, which improves security
- ✓ As we discussed earlier, if a data member is declared "private", then it can only be accessed within the same class. No outside class can access data member (variable) of other class.
- ✓ However, if you need to access these variables, you have to use **public "getter" and "setter"** methods.



NEXT

public static void main(String[] args)

```
class StaticMethod {
        public static void main(String[] args)
        {
                System.out.println("I am a Static");
        }
}
```

Static methods

- Can be called without creating an object of class.
- They are referenced by the **class name itself** or reference to the Object of that

```
public static void geek(String name)
 {
 // code to be executed....
 }
// Must have static modifier in their
declaration.
// Return type can be int, float,
String or user defined data type
```

Important Points:

✓ Static method(s) are associated to the class in which they reside
  - ➢ i.e. they can be called even without creating an instance of the class i.e ClassName.methodName(args).

✓ They are designed with aim to be shared among all Objects created from the same class.

✓ Static methods can not be overridden. But can be overloaded since they are resolved using **static binding** by compiler at compile time.

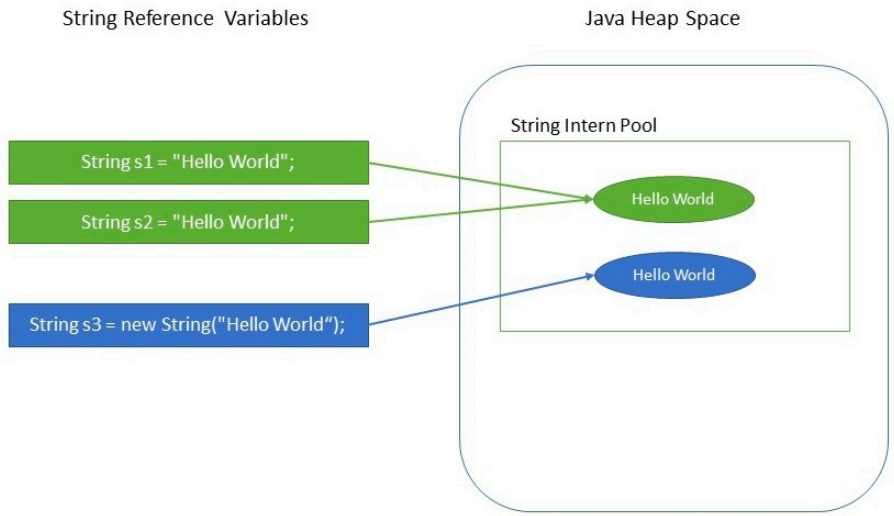
Static Class

Why Java is not a Pure Object Oriented Language?

Primitive Data Type

The static keyword

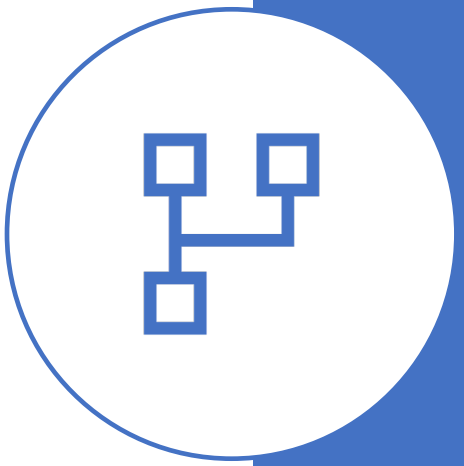
Strings in Java



- String vs StringBuilder vs StringBuffer
- Integer to String Conversions
- String to Integer– parseInt()
- Swap two Strings without using third variable
- Searching characters and substring in a String
- Compare two Strings in Java
- Reverse a string in Java (5 Different Ways)
- Remove Leading Zeros From String in Java
- Trim (Remove leading and trailing spaces) a string
- Counting number of lines, words, characters and paragraphs in a text file using Java
- Check if a string contains only alphabets in Java using Lambda expression
- Remove elements from a List that satisfy given predicate in Java
- Check if a string contains only alphabets in Java using ASCII values
- Check if a string contains only alphabets in Java using Regex

Arrays in Java

- Arrays in Java(Practice)
- Default Array values
- Util Arrays Class (Contains utility functions for Arrays)
- Interesting facts about Array assignment in Java
- Array **IndexOutofbounds** Exception
- Array vs ArrayList in Java
- **Implementations**:
- Compare two arrays
- ArrayList to Array Conversion
- Merge arrays into a new object array in Java

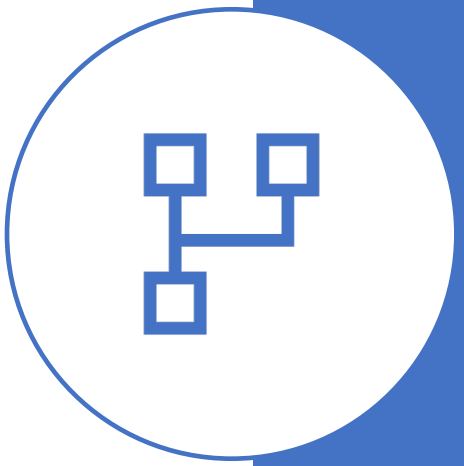


Stream in Java

- Java Stream

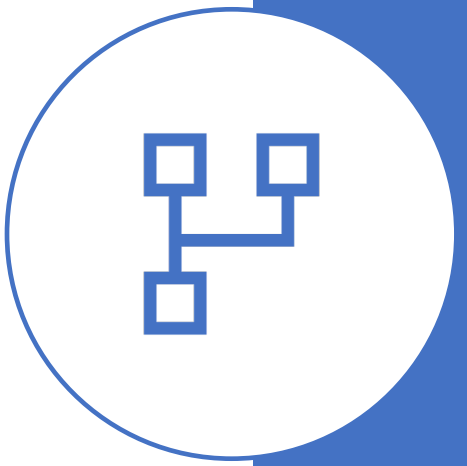
Implementations:

- Features
- 10 Ways to Create a Stream in Java
- How to get ArrayList from Stream in Java 8
- Stream.of() vs. Arrays.stream()



Stream Features

- A stream is not a data structure and does not store elements. Collections, Arrays or I/O Channels is where it takes the input from.
- The source of the stream remains unmodified after operations are performed on it. For example, filtering a stream simply produces a new stream without the filtered elements, instead of modifying the original stream.
- Aggregation operations such as filter, reduce, match, find, etc are supported by stream.



Methods in Java

- Methods
- Parameters passing
- Returning Multiple values
- Throwable fillInStackTrace() method in Java
- Method Overloading
- Different ways of Method Overloading in Java
- Overriding equals method
- Overriding toString() method
- Private and final methods
- Java is Strictly Pass by Value
- Clone() method
- Remote Method Invocation
- Default Methods
- Passing and Returning Objects in Java
- Date after() method in Java
- System.exit() method

Exception Handling

- Exceptions
- OutOfMemoryError Exception
- 3 Different ways to print Exception messages in Java
- flow control in try-catch-finally
- Types of Exceptions
- Catching base and derived classes as exceptions
- Checked vs Unchecked Exceptions
- Throw and Throws
- User-defined Custom Exception
- Infinity or Exception?
- Multicatch
- Chained Exceptions
- Null Pointer Exception
- Output of Java program | Set 12(Exception Handling)

Collection in Java

- Collection:
- Collections Class in Java
- Enumeration, Iterators and ListIterators
- Convert an Iterable to Collection in Java
- Using Iterators
- Iterator vs Foreach
- Types of iterator
- Creating Sequential Stream from an Iterator in Java
- Implementations:
- Output of Java Program | Set 13(Collections)
- Immutable List in Java (Guava)
- java.util.Concurrent

Collection Interview FAQ's

- Vector vs ArrayList
- ArrayList vs LinkedList
- Comparable vs Comparator
- Differences between TreeMap, HashMap and LinkedHashMap
- HashMap vs HashTable
- Hashmap vs WeakHashMap in Java
- How to Synchronize ArrayList in Java
- ArrayList and LinkedList remove() methods
- How to Remove an element from ArrayList

Garbage Collection

- Garbage Collection
- How to make object eligible for garbage collection in Java?
- Automatic Resource Management
- Output of Java programs | Set 10 (Garbage Collection)
- Iterator vs Collection in Java

File Handling

- File class
- Ways of Reading a text file in Java
- file permissions in java
- Moving a file from one directory to another using Java
- Copying file using FileStreams
- Delete a file using Java
- Java program to delete duplicate lines in text file
- Java program to merge two files alternatively into third file
- Java program to List all files in a directory and nested sub-directories | Recursive approach
- Java program to delete certain text from a file
- Check if a File is hidden in Java
- Redirecting System.out.println() output to a file