# JAVA

Java is OOP programming language. Java is good scalable and platform independent

##### **Naming of Java. What is J2?**

* Java 1.0 was developer and released, but had a lot of bugs
* Java 1.1 and Java 1.2 were released
* Java 1.2 was so successful and recognized by people that they decided that previously was “just Java” and now it is “Java2 = J2”. So it means Java2 version 8 = Java 1.8 – it is the same

It was developed by the company Sun Microsystems in 1995. It is a specification and includes topics like Servlet, JSP, Web Services, EJB, JPA

Java code is compiled into bytecode by compiler and can be run on multiple platforms

**JVM (Java Virtual Machine)** – is an abstract machine and it does not physically exist. It is a specification that provides runtime environment where bytecode will be executed. JVM are available for many platforms and OS. Therefore Java is platform independent

**JRE (Java Runtime Environment)** – it is used to provide runtime environment. It is the implementation of JVM

**JDK (Java Development Kit)** – is a software development environment which is used to develop Java applications. It contains JRE + development tools

**JDKs:**

* ME (Micro Edition) – for appliances, mob devices
* SE (Standard Edition)
* EE (Enterprise Edition)

# JAVA OOP

##### **History of programming:**

1. Algorithms
2. Procedural programming
3. OOP paradigm
4. Others: functional programming, aspect programming

**OOP:**

Purpose or OOP languages is to implement real word entities

OOP language over procedural

* It’s easy to develop and support applications over procedural language
* OOP provides data hiding. Whereas in procedural language global data can be accessed anywhere
* Make code reusable
* It is much easy to develop app by a few developers compared to procedural langauge

**CLASS** is blue print from which you can create [objects]

Class is logical entity

**OBJECT** is runtime entity that has state and behavior. Object is created from the class or instance of class. Everything that surrounds us is ab object (pen, table, chair, car)

Object is data and methods that process those data

For example, dog is an object.

* It has state - color, name, whiskers
* It has behavior – barking, eating, wagging tail

Object has address to the memory and therefore it takes up some space

Object is physical entity

Object has:

* **state** - represent data (value) of an object
* **behavior** – represents functionality (deposit, withdraw)
* **identity** – is ID. It is not visible to external users. It’s used internally by JVM to identify each object uniquely

[**new**] keyword is used to allocate memory at runtime. All objects get memory in [Heap memory] are

**Declaration -> Instantiation-> Initialization**

**1.Declaration** assign type to the variable

Person p;

**2.Instantiation** means to creating new object. It means allocation memory at runtime in [Heap] memory ] area

Person p = new Person();

Person p = new Person(“Alex”);

**3.Initialization** – passing parameters to constructor to define (=*storing data in Object*)

public class Person

{

private String name;

private String color;

public Person(String name)

{…}

public Person(String name, String color)

{…}

}

# INITIALIZE OBJECT

There are 3 ways to initialize object:

1. **by reference variable**



1. **by method**



1. **by constructor**



# INSTANTIATE OBJECT

It means allocation memory at runtime in [Heap] memory

There are 5 ways to do it:

* By [new ] keyword
* By [newInstance()] method
* By [clone] method
* By deserialization
* By factory method

# CONSTRUCTOR

**Constructor** is - a special method that does not have return method and must have the same name as class.

Constructor can not be:

* status
* final
* synchronized

Constructor *overloading* is allowed in Java. Overloading is just have constructor with different parameters

|  |  |
| --- | --- |
| CONSTRUCTOR | MEHTOD |
| *Constructor is used to initialize the state of Object* | Method is to make some behavior |
| It does not have return type | It must have return type |
| Constructor is invoked implicitly | method is invoked explicitly |
| Java provides default constructor | Method must be developed |
| Constructor name must match with class name | Any name |

public class Student

{

public Student()

{…}

public Student(String name)

{…}

public Student(int id, String name)

{…}

}

# STATIC

Static relates to class than instance.

Static can be

* **variable** - it is used for cases when variables is used across all object. For example, SK, company name, university name. Static variable **gets memory only once**.
* **method** -
* static block – this block executed when class is loaded in the memory. It’s used to initialize static data
* nested class -

note: is it possible to execute java program without main method? It was possible in static bloch till JGK 1.6. After it’s not possible

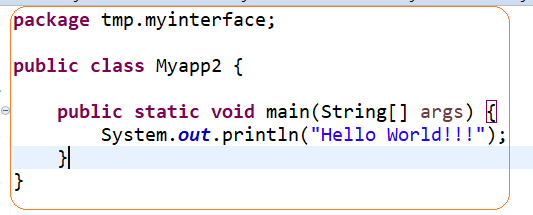
##### STATIC NAME CONVENTION

# FINAL

If class marked as final you cannot make inheritance. It is usually by done for security purposes

By name convention all constants must be in upper case: DAY\_OF\_THE\_DAY, SK

# RUN MAIN METHOD



**public** – access modifier which provides visibility. It means it is visible to all

**static** – is a keyword. There is no need to invoke object of class. So, it saves memory

**void** – return type

**main** - starting point of program

**String[] args** – is used for command line argument

# STATE OF OBJECT

**State of object** is a value of all fields of object

# ENCAPSULATION

**ENCAPSULATION** - means when internal state of object can be changed only by this object

# INHERITANCE

**INHERITANCE** – is OOP concept. When one object acquires all the properties and behaviors of parent class it’s called inheritance.

It’s used to achieve [runtime polymorphism]

# POLYMORFISM

**POLYMORFISM** is when task can be performed in different ways

**POLYMORFISM** is the ability of object takes different forms

Polymorphism:

* static binding
* dynamic binding

##### **overriding**

**overriding** -

# ADBSTRACTION

**ABSTRACTION** is hiding details and showing only essential functionality. Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details.

Example1: Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car but he does not know about how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car. This is what abstraction is.

* *Abstraction can by achieved by interface and abstract class*
* **Abstract class can not be directly instantiated (with the new operator).**
* **Abstract method** is a method that is declared without an implementation.
* If class has at least one abstract method it has to be abstract

Pros of abstraction:

* Avoids code duplication and increases reusability.
* It reduces the complexity of viewing the things.
* Helps to increase security of an application or program as only important details are provided to the user.
* It I used for hierarchy
* It is used to achieve polymorphism (overriding methods)

**Abstract method** – method that has only signature (without body)

# ABSTRACTION vs ENCAPSULATION

Differences:

1. **Abstraction is implementation hiding, while encapsulation is data hiding**.
2. Abstraction is implemented using abstract class and interface while encapsulation is implemented using access modifiers(private, )

# INTERFACE

##### **Introduction**

Prior to java 8, interface in java can only have abstract methods. All the methods of interfaces are public & abstract by default. Java 8 allows the interfaces to have default and static methods. The reason we have default methods in interfaces is to allow the developers to add new methods to the interfaces without affecting the classes that implements these interfaces.

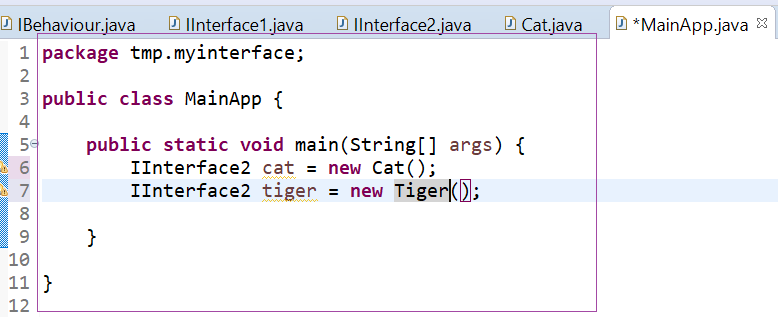
**Interface** – is a contract. I t describes what behavior class will have

**Why need interface**:

1. It describes what behavior class will have. For example, animal will have methods – sleep(), hunt(), eat(), but each class will have own implementation
2. Interfaces are used to provide **loose coupling**.
3. Java does not support multiple inheritances but we can achieve the effect of multiple inheritances using interfaces. Java does not support multiple inheritances but we can achieve the effect of multiple inheritances using interfaces

**Tagged or Marker interface**

Tagged or Marker interface – is interface without methods. These methods are used to tag or marking a class. So that you can determine whether someclass is a child of those classes.



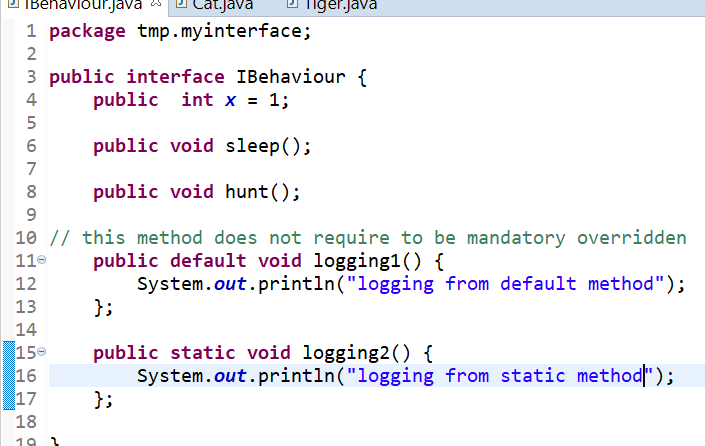
Example of marker interfaces are **Serializable, Cloneable.**

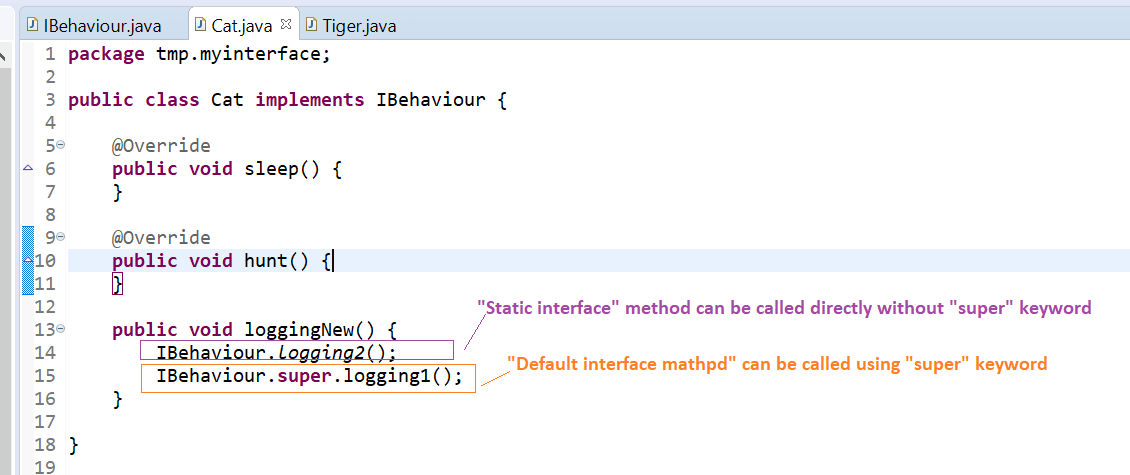
**Default interface methods**

For example, if several classes such as A, B, C and D implements an interface XYZInterface then if we add a new method to the XYZInterface, we have to change the code in all the classes(A, B, C and D) that implements this interface. In this example we have only four classes that implements the interface which we want to change but imagine if there are hundreds of classes implementing an interface then it would be almost impossible to change the code in all those classes. This is why in java 8, we have a new concept “default methods”. These methods can be added to any existing interface and we do not need to implement these methods in the implementation classes mandatorily, thus we can add these default methods to existing interfaces without breaking the code.

**Static interface methods**

Static methods in interfaces are similar to the default methods except that we cannot override these methods in the classes that implements these interfaces.Static methods in interfaces are similar to the default methods except that we cannot override these methods in the classes that implements these interfaces.





# ABSTRACTION vs INTERFACE

Difference:

* Abstract class can have constructor while Interface can not
* Interface provides a full abstraction while abstract class partial
* A Java interface can be implemented using keyword “implements” and abstract class can be extended using keyword “extends”. A Java interface can be implemented using keyword “implements” and abstract class can be extended using keyword “extends”.
* Java does not support multiple inheritances but we can achieve the effect of multiple inheritances using interfacesJava does not support multiple inheritances but we can achieve the effect of multiple inheritances using interfaces

# COUPLING

**COUPLING** refers to dependency to another class. You can use interface to achieve weakly coupling, because it does not have concrete implementation.

# COHESION

Weakly cohesive method will split the task into separate parts

# ASSOCIATION

**ASSOCIATION** represents relationship between objects:

* 1:1
* 1:n
* n:1
* n:n

# AGGREGATION

**AGGREGATION** (**HAS-A**) is a way to achieve association.

**AGGREGATION** (**HAS-A**) represents the relationship where one object contains others as a part of its own state. It represents a **weak relationship** between objects

represents re

# COMPOSITION

**COMPOSITION**(**IS-A**) is also way to represents the relationship where one object contains others as a part of its own state. It represents a **strong relationship** between objects

# [this] keyword

* [this] can be used to refer to instance variable of current class
* [this] can be used to invoke current class method
* [this] can be used to invoke current constructor method
* [this] can be used to pass argument to the method

Facts:

* You cannot use [this] for static methods

1.[this] can be used to refer to instance variable of current class



2.[this] can be used to invoke current class method



3.[this] can be used to invoke current constructor method



4.[this] can be used to pass argument to the method



|  |  |
| --- | --- |
| **HOW ACHIEVE CONCEPT** | **DESC** |
| abstraction | Using abstract class and inheritance |
| runtime polymorphism | Inheritance (reference variables) |
| STRING |  |
|  |  |

# METHOD SIGNATURE

**Method signature** - consists of method name and parameter lists

* Method signature does not include the return type of the method. A class cannot have two methods with same signature. If we try to declare two methods with same signature you will get a compile time error.

# DATA TYPE

There are 2 types of datatypes:

* Primitive
* Object

##### **objects**

My = new My(); new My() does not have object it has reference to My

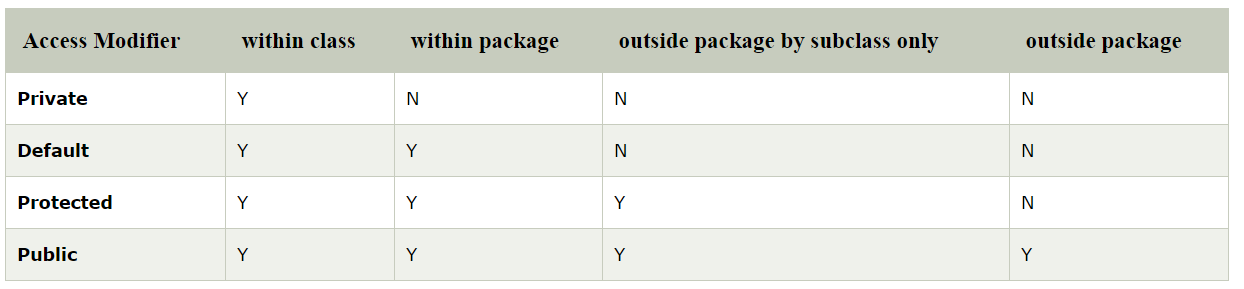
##### **primitives**

* For primitive types exist [obvertka]. For example, for [int] exists [Integer]
* You can not use primitives for collections

# BOXING and UNBOXING

# ACCESS MODIFIERS

* **private** -
* **default = package local** -
* **protected** – all childs of parent class will have access
* **public** -



# LINKS

Links:

<https://www.javatpoint.com/this-keyword>

* equals:

<http://users.csc.calpoly.edu/~gfisher/classes/102/info/howToOverrideEquals.html>

* loose coupling

<https://www.interviewsansar.com/2018/03/24/loose-coupling-and-tight-coupling-in-java/>

* generics

<https://www.baeldung.com/java-generics-interview-questions>

<https://www.journaldev.com/1663/java-generics-example-method-class-interface>

<https://howtodoinjava.com/java/generics/complete-java-generics-tutorial/amp/>

* web

<https://www.javatpoint.com/get-vs-post>

(!!!!)

<https://www.restapitutorial.com/lessons/httpmethods.html>

<https://www.journaldev.com/1854/java-web-application-tutorial-for-beginners>

<https://www.journaldev.com/2513/tomcat-datasource-jndi-example-java>

<https://www.journaldev.com/1854/java-web-application-tutorial-for-beginners>

* jsp

<https://www.tutorialspoint.com/jsp/>

* Rest-api

<https://zapier.com/learn/apis/>