

OOP CONCEPTS IN JAVA

PANDORA (ANDROID)

2

Java Basics

Basic Syntax of Java and differences from C++

Source Structure

3

- ❑ Java is strictly Object-Oriented
- ❑ Except primitive data types, everything is a Class
- ❑ Each java file Source.java represents a Class
- ❑ Divided into packages

Java Classes

4

- ❑ Fundamental concept of Java
- ❑ Describes data objects, and what we can do with them
- ❑ Contains data (objects) and logic (methods)
- ❑ Every object must instance of a class
- ❑ Every method must belong to a class

Java Classes

5

```
public class Person {  
    private String firstName;  
    private String lastName;  
  
    public int age;  
  
    public String getFullName() {  
        return firstName + lastName;  
    }  
}
```

Comments

6

```
public class Person {  
    private String firstName;  
    private String lastName;  
    //This is the person's age  
    public int age;  
    /* This function  
    shows full name of person  
    */  
    public String getFullName() {  
        return firstName + lastName;  
    }  
}
```

Scopes

7

```
void someFunction () {  
    int a = 10;  
    {  
        int a = 20; //Invalid in Java, works in C++  
    }  
}
```

Arrays

8

- Arrays are also objects
- Java has no pointers
- Arrays are garbage-collected
- Memory is allocated dynamically

Modifiers

9

- A 'control' keyword applicable to methods and variables.
- Access modifiers : For visibility
- Non-access modifiers: Other functionalities

Access Modifiers

10

- ☐ package
- ☐ private
- ☐ protected
- ☐ public

Non-Access Modifiers

11

- static
 - ▣ Part of class or part of object
- final
 - ▣ Can be reimplemented (or data changed) or not
- abstract
 - ▣ Is fully implemented or not
- synchronized
 - ▣ Thread safe or not

Static

12

- ❑ Objects or methods
- ❑ Independent of any object of that class
- ❑ Cannot use non-static members inside static members
- ❑ Static methods can be called as `ClassName.staticFunction()`

Final

13

- ❑ Makes the reference of first initialization of the object constant
- ❑ The reference cannot change
- ❑ Internal data of that object can still change (difference from const in C++)

Constructors

14

- ❑ Used for creating instances of class (objects)
- ❑ One or multiple constructors (overloaded)
- ❑ Name is same as class name
- ❑ A default empty constructor exists if you don't make one
- ❑ By default recursively calls 0-argument constructors of superclass.

Creating objects

15

- Unlike C++, calling constructor is necessary
- `Foo a; //This will be null`
- `Foo a = new Foo(); //This creates reference to obj`

OOPs Concepts

16

- ❑ Encapsulation
- ❑ Inheritance
- ❑ Polymorphism
- ❑ Abstraction

Encapsulation

17

- ❑ Binding code and data into single unit (class)
- ❑ Can change data and/or its implementation in one unit, without breaking other units.
- ❑ Keeps data and code safe from external interference

Inheritance

18

- ❑ One class can extend the data and/or functionality of existing class.
- ❑ Creating a class 'based on' another class.
- ❑ Mango inherits from Fruit. i.e. Mango is a type of Fruit. Mango has all properties of a Fruit, plus some of its own.

Polymorphism (functions)

19

- One thing (usually a method) taking multiple forms
 - ▣ `area(int r)` – area of circle
 - ▣ `area(int w, int h)` – area of rectangle
 - ▣ If we call `area(10)`, it will find area of circle.
 - ▣ If we call `area(5,8)`, it will find area of rectangle.

Polymorphism (reference)

20

- A reference to a subclass can be used as object of superclass.
- `Fruit f = new Mango(); //This works`
- `f.getFruitColor(); // This works`
- `f.getMangoType(); //Won't work`

Java – Extending Classes

21

- One class can have **only one** super class.
- Class Mango extends Fruit { }
- Java arranges Classes in hierarchy
 - ▣ Food -> Fruit -> Mango
- When calling member function of same signature, closest in class hierarchy is called.
 - ▣ Mango.getType() is preferred over Fruit.getType()

Java - Interfaces

22

- ❑ Purely abstract. Has no implementations.
- ❑ All fields are public static final
- ❑ All methods are public abstract
- ❑ Interfaces are *implemented* by a class.
- ❑ If not all methods are defined, then the implementing class will have to be abstract.
- ❑ An interface is a 'contract' that has to be fulfilled if it is implemented

Java - Interfaces

23

```
public interface FruitInterface {  
    boolean isEdible = true;  
    public String getOrigin();  
}
```

```
public class Mango extends Fruit implements FruitInterface  
{  
    @Override  
    public String getOrigin() {  
        return "Lucknow";  
    }  
}
```

Java - Interfaces

24

- ❑ Interfaces cannot be *instantiated*
- ❑ Has no constructors
- ❑ All methods are abstract
- ❑ Cannot have non-static fields
- ❑ An interface extend multiple interfaces

Java - Generics

25

- ❑ Enable *types as parameters*
- ❑ Reuse same class or methods with different data types.
- ❑ Eliminate typecastings.
- ❑ Implement generic data structures and algorithms (Graphs, Trees, Lists; shortest path, flatten, sort)

Java - Generics

26

```
public class Entry {  
    String key;  
    String value;  
  
    public String getKey() {  
        return key;  
    }  
  
    public void setKey(String key) {  
        this.key = key;  
    }  
  
    public String getValue() {  
        return value;  
    }  
  
    public void setValue(String value) {  
        this.value = value;  
    }  
}
```

Java - Generics

27

```
public class Entry <T> {  
    T key;  
    T value;  
  
    public T getKey() {  
        return key;  
    }  
  
    public void setKey(T key) {  
        this.key = key;  
    }  
  
    public T getValue() {  
        return value;  
    }  
  
    public void setValue(T value) {  
        this.value = value;  
    }  
}
```

Java - Generics

28

```
public class Entry <K,V> {  
    K key;  
    V value;  
  
    public K getKey() {  
        return key;  
    }  
  
    public void setKey(K key) {  
        this.key = key;  
    }  
  
    public V getValue() {  
        return value;  
    }  
  
    public void setValue(V value) {  
        this.value = value;  
    }  
}
```

Java - Exceptions

29

- ❑ An event that disrupts the normal flow of the program
- ❑ Exceptions are powerful mechanisms to handle runtime errors
- ❑ There are many inbuilt exceptions
 - ❑ NullPointerException
 - ❑ ArrayOutOfBoundsException
 - ❑ IOException

Java - Exceptions

30

- We can define our own exceptions
- Types of exceptions –
 - ▣ Checked Exception: Expected problems
 - ▣ Errors: Unexpected, externally induced
 - ▣ Unchecked Exception: Faulty logic in code

Java – Exception Handling

31

- Try:
 - ▣ The code we attempt to execute goes here
- Catch:
 - ▣ Can be multiple catch blocks for different exceptions
 - ▣ Take remedial action when exception has occurred
- Finally:
 - ▣ This block gets called, regardless of exception occurred or not.