Here's a solid breakdown of Java basics, focusing especially on:

1. OOP Concepts
2. Interfaces
3. Exceptions
4. Collections

## 1. **Java OOP (Object-Oriented Programming) Basics**

Java is fully object-oriented (except for primitive types). Key OOP pillars:

### a. **Class and Object**

* Class: Blueprint (e.g., Car)
* Object: Instance of class (e.g., Car myCar = new Car();)

### b. **Encapsulation**

* Hiding internal state using private fields.
* Access via getters/setters.

class Person {

private String name;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

### c. **Inheritance**

* Reusing code from parent class using extends.

class Animal {

void sound() {

System.out.println("Animal makes sound");

}

}

class Dog extends Animal {

void bark() {

System.out.println("Dog barks");

}

}

### d. **Polymorphism**

* Same method name, different behavior.

Compile-time (Method Overloading):

class Math {

int add(int a, int b) { return a + b; }

double add(double a, double b) { return a + b; }

}

Runtime (Method Overriding):

class Animal {

void sound() { System.out.println("Animal sound"); }

}

class Cat extends Animal {

void sound() { System.out.println("Meow"); }

}

### e. **Abstraction**

* Hiding complex logic and showing only essential features.
* Achieved via abstract classes or interfaces.

## 2. **Interfaces**

An interface defines what a class should do, not how.

interface Drivable {

void drive();

}

class Car implements Drivable {

public void drive() {

System.out.println("Car is driving");

}

}

* A class can implement multiple interfaces.
* Interfaces support default and static methods (Java 8+).

interface A {

default void show() {

System.out.println("Default method in A");

}

}

## 3. **Exceptions**

### a. **Types**

* Checked: Must be caught or declared (IOException)
* Unchecked: Runtime errors (NullPointerException)

### b. **Try-Catch-Finally**

try {

int x = 5 / 0;

} catch (ArithmeticException e) {

System.out.println("Divide by zero");

} finally {

System.out.println("Always runs");

}

### c. **Throw vs Throws**

// 'throw' - throw an exception

throw new IllegalArgumentException("Invalid");

// 'throws' - declare method might throw

public void readFile() throws IOException {

// ...

}

### d. **Custom Exception**

class MyException extends Exception {

public MyException(String message) {

super(message);

}

}

## 4. **Collections Framework**

Java Collections = set of classes for data structures like List, Set, Map, Queue.

### a. **List**

* Ordered, allows duplicates
* ArrayList, LinkedList

List<String> names = new ArrayList<>();

names.add("Alice");

### b. **Set**

* No duplicates
* HashSet, TreeSet

Set<Integer> ids = new HashSet<>();

ids.add(101);

### c. **Map**

* Key-value pairs
* HashMap, TreeMap

Map<String, Integer> ageMap = new HashMap<>();

ageMap.put("John", 25);

### d. **Queue**

* FIFO structure
* LinkedList, PriorityQueue

Queue<String> queue = new LinkedList<>();

queue.add("Task 1");

### ✅ Summary

| Concept | Keyword/Example | Purpose |
| --- | --- | --- |
| Class/Object | class, new | Blueprint/instance |
| Encapsulation | private, getters | Data hiding |
| Inheritance | extends | Reuse code |
| Polymorphism | overload/override | Multiple behaviors |
| Interface | implements | Contract for behavior |
| Exceptions | try-catch-finally | Handle errors |
| Collections | List, Map, etc. | Data storage & manipulation |