1. Inheritance

Inheritance

Inheritance allows a class (subclass) to inherit fields and methods from another class (superclass). It promotes code reuse and supports polymorphism.

- Syntax:

```
public class ChildClass extends ParentClass { }
```

- Example:

```
abstract class Account {
    protected double balance;
    public abstract double calculateInterest();
}
class SavingsAccount extends Account {
    public SavingsAccount(double balance) {
        this.balance = balance;
    }
    @Override
    public double calculateInterest() {
        return balance * 0.05;
    }
}
```

- super() usage:

Used in the constructor of a child class to call the parent constructor.

- Abstract Classes:

You cannot instantiate an abstract class. Subclasses must implement abstract methods.

- Key Rule:

Account acc = new SavingsAccount(1000); // Polymorphism

acc.calculateInterest(); // Calls overridden method in SavingsAccount

2. Polymorphism

Polymorphism

Polymorphism allows us to use a superclass reference to refer to a subclass object.

- Example:

Account acc = new ChequeAccount("CH123", "Thabo", 1000);

System.out.println(acc.calculateInterest());

Even though the type is Account, the method from ChequeAccount runs.

- Why it's powerful:

It lets us write general code that works with any subclass of Account.

3. Abstraction

Abstraction

Abstraction hides complex details and shows only essential features.

- Abstract Class Example:

```
abstract class Account {
    public abstract double calculateInterest();
}
```

Subclasses must implement calculateInterest().

4. Encapsulation

Encapsulation

Encapsulation means keeping class variables private and exposing access via getters/setters.

- Example:

```
class Player {
    private int score;
    public void setScore(int score) {
        if (score < 0) throw new IllegalArgumentException("Invalid score");
        this.score = score;
    }
    public int getScore() {
        return score;
    }
}
```

5. Data Validation & Exceptions

Data Validation and Exceptions

Use setters to validate input and throw exceptions if needed.

- Example:

```
public void setBalance(double balance) {
    if (balance < 0) throw new IllegalArgumentException("Balance cannot be negative");
    this.balance = balance;
}
```

6. JUnit Testing

JUnit Testing

JUnit is a framework used for unit testing Java code.

- Example:

```
@Test

public void testCalculateInterest() {

    SavingsAccount acc = new SavingsAccount("S123", "Lerato", 10000);

    assertEquals(500, acc.calculateInterest(), 0.01);

}
```

- Test Invalid Data:

```
@Test(expected = IllegalArgumentException.class)

public void testInvalidBalance() {

    new SavingsAccount("S100", "Zola", -100);

}
```