```java
// Runtime Polymorphism
// Dynamic Binding
// Late Binding
// Dynamic method dispatch possible for overridden methods. But this logic not applicable to variables.
// To use Dynamic method dispatch, child class method must need to be in the parent class.
// Object Casting possible - Up Casting (Implicit) and Down Casting (Explicit)
// Use of "instanceof" operator - to check if object is of selected class or not
//--> Syntax: objectname instanceof ClassName Ex: a instanceof A
// instanceof returns boolean - true or false


public class DynamicMethodDispatch {
    public static void main(String[] args) {
        A a = new A(); // Valid
        A a1 = new B(); // Valid Implicit UpCasting
        B b = new B(); // Valid
        // B b1 = a; // Invalid  // C.E. : Type mismatch: cannot convert from A to B
        // B b1 = (B) a; // C.E. : java.lang.ClassCastException: class A cannot be cast to class B
        // B b2 = a1; // Type mismatch: cannot convert from A to B
        B b2 = (B) a1; // Valid Explicit DownCasting
        a.m1();
        a1.m1();
        a1.m2();
        b.m2();
        b2.m1();
        b2.m2();
        System.out.println(b instanceof A); // true
        System.out.println(a instanceof B); // false
        System.out.println(a instanceof A); // true
        System.out.println(a1 instanceof A); // true
        System.out.println(b2 instanceof A); // true
    }
}

class A {
    void m1() {
        System.out.println("Class A m1 Method");
    }

    void m2() {
        System.out.println("Class A m2 method");
    }
}

class B extends A {
    void m2() {
        System.out.println("Class B m2 Method");
    }
}
```

```java
/*
Abstraction:
    --> Hiding internal Implementation
    --> Keyword: abstract (Can be used with class and/or methods)
    --> abstract class cannot be instantiated
    --> to use abstact class, it need to be extended.
    --> After extending, use child class object and parent class
    reference or child class object and reference to use abstract class
    data

Abstract Methods:

    --> Only declare abstract method in abstract class, never perform
    method implementation
    --> Abstract methods can not have body
    --> Class must be abstract if class have atleast one abstract method.
    --> Abstract class can have any method abstract or non abstract.
    --> final and abstract combination never possible (Illegal
    combination)
    --> abstract method can not be declared as static
    --> All abstract methods must be implemented in child class.
    --> If child class implements all abstract methods, then parent
    class can be implemented.
    --> Abstract cass can have constructor.
    --> Static variable of abstract class can be called using abstract
    classname or Child object.
    --> Main method can be considered in abstract class also.

 */

abstract public class Abstract1 {
    void m1() {
        System.out.println("Regular m1 method");
    }

    abstract void m11();

    abstract void m111();
}

class ImpAbstract1 extends Abstract1 {
    // All abstract methods must need to implement in sub class
     void m11() {
        System.out.println("Abstract Implemented m11 method");
    }

    public void m111() {
        System.out.println("Abstract  Implemented m111 method");
    }
}

class Main {
    public static void main(String[] args) {
        Abstract1 obj1 = new ImpAbstract1(); // Valid
        // ImpAbstract1 obj2 = new ImpAbstract1(); // Valid
```

```
50              obj1.m1();
51              obj1.m11();
52              obj1.m111();
53          }
54      }
```

```java
abstract public class Abstract2 {
    void m2() {
        System.out.println("Regular m2 method");
    }

    abstract void m22();

    abstract void m222();

}

abstract class ImpAbstract2 extends Abstract2 {
    public void m22() {
        System.out.println("Abstract Implemented m22 method");
    }
}

class ImpAbstract22 extends ImpAbstract2 {
    public void m222() {
        System.out.println("Abstract Implemented m222 method");
    }
}

    class Main1 {
        public static void main(String[] args) {
            ImpAbstract22 obj1 = new ImpAbstract22();
            obj1.m2();
            obj1.m22();
            obj1.m222();
        }
    }
```

```java
// Abstract class can have constructor
public abstract class Abstract3 {
    int num;
    String fname;

    public Abstract3(int num, String fname) {
        this.num = num;
        this.fname = fname;
    }

    public Abstract3() {
        System.out.println("Default constructor Called");
    }

    void display() {
        System.out.println("Abstract 3 Display method Called");
    }
}

class Main2 extends Abstract3 {

    String lname;

    Main2(int num, String fname, String lname) {
        super(num, fname);
        this.lname = lname;
    }

    Main2() {
        System.out.println("Main Default Constructor Called");
    }

    void display() {
        System.out.println("Main 2 Display method Called");
    }
    public static void main(String[] args) {
        Abstract3 a = new Main2(1, "abc", "xyz");
        a.display();

    }
}
```