Object Oriented Programming

ඔබ්ජෙක්ට්-ඔරියන්ටඩ් ප්‍රෝග්‍රැමින් (OOP) යනු ප්‍රෝග්‍රැමින් හි එක් ප්‍රධාන මොඩලයකි. එය ඔබ්ජෙක්ට් (Objects) මත පදනම් වූ ප්‍රෝග්‍රැමින් ක්‍රමවේදයකි. OOP හි මූලික සංකල්ප කිහිපයක් තිබේ, ඒවා නම්:

1. ඔබ්ජෙක්ට් (Object) : ඔබ්ජෙක්ට් යනු තත්ත්ව (attributes) සහ හැසිරීම් (methods) සහිත සංකල්පනීය හෝ භෞතික ඒකකයකි. උදාහරණයක් ලෙස, "කාර්" යනු ඔබ්ජෙක්ට් එකකි. එහි තත්ත්ව ලෙස වර්ණය, වේගය, ඉන්ධන මට්ටම් ආදිය ද, හැසිරීම් ලෙස ධාවනය කිරීම, නවතිනවා, ඉන්ධන පිරවීම ආදිය දැක්විය හැක.

2. ක්ලාස් (Class) : ක්ලාස් යනු ඔබ්ජෙක්ට් සඳහා නිර්මාණ රටාවකි හෝ බ්ලූප්‍රින්ට් එකකි. ක්ලාස් එකක් භාවිතා කරමින් ඔබට බහුලව ඔබ්ජෙක්ට් නිර්මාණය කළ හැක. උදාහරණයක් ලෙස, "කාර්" ක්ලාස් එකක් භාවිතා කරමින් විවිධ කාර් ඔබ්ජෙක්ට් නිර්මාණය කළ හැක.

3. ඉන්හෙරිටන්ස් (Inheritance) : ඉන්හෙරිටන්ස් යනු එක් ක්ලාස් එකක් තවත් ක්ලාස් එකකින් ගුණාංග උරුම කර ගැනීමේ ක්‍රමයයි. උදාහරණයක් ලෙස, "වාහන" ක්ලාස් එකක් තිබේ නම්, "කාර්" ක්ලාස් එක "වාහන" ක්ලාස් එකෙන් ඉන්හෙරිට් කරගත හැක.

4. පොලිමෝපිසම් (Polymorphism) : පොලිමෝපිසම් යනු එකම ක්‍රමයක් විවිධ ආකාරවලින් ක්‍රියාත්මක වීමයි. උදාහරණයක් ලෙස, "වාහන" ක්ලාස් එකේ "ධාවනය" ක්‍රමයක් තිබේ නම්, "කාර්" සහ "මෝටර්සයිකල්" වල එම ක්‍රමය විවිධ ආකාරවලින් ක්‍රියාත්මක විය හැක.

5. එන්කැප්සුලේෂන් (Encapsulation) : එන්කැප්සුලේෂන් යනු දත්ත සහ ක්‍රම එක් ඒකකයක් තුළ ඇසුරුම් කිරීමයි. එය දත්ත සුරක්ෂිත කිරීම සහ කේතය පහසුවෙන් පාලනය කිරීමට උපකාරී වේ.

6. ඇබ්ස්ට්‍රැක්ෂන් (Abstraction) : ඇබ්ස්ට්‍රැක්ෂන් යනු සංකීර්ණ තත්ත්වයන් සරල කිරීමේ ක්‍රමයකි. එය අවශ්‍ය තොරතුරු පමණක් පෙන්වන අතර, අනවශ්‍ය තොරතුරු සඟවයි.

OOP භාවිතා කිරීමෙන් ප්‍රෝග්‍රැම් වල සංවිධානය, නැවත භාවිතය, සහ පාලනය පහසු වේ. එය මෘදුකාංග සංවර්ධනයේදී බහුලව භාවිතා වන මොඩලයකි.

ජාවා (JAVA) හැඳින්වීම

ජාවා යනු ඉහළ මට්ටමේ, ඔබ්ජෙක්ට්-ඔරියන්ටඩ් (Object-Oriented) ප්‍රෝග්‍රැමින් භාෂාවකි. එය 1995 දී සන් මයික්‍රොසිස්ටම්ස් (Sun Microsystems) විසින් නිර්මාණය කරන ලද අතර, වර්තමානයේදී එය ඔරකල් කෝපරේෂන් (Oracle Corporation) විසින් පාලනය කරනු ලබයි. ජාවා භාවිතා කරමින් වෙබ් යෙදුම්, මොබයිල් යෙදුම්, ඩෙස්ක්ටොප් යෙදුම්, සර්වර් යෙදුම්, සහ විශාල පරිමාණයේ පද්ධති සංවර්ධනය කළ හැක.

ජාවාවේ ප්‍රධාන ලක්ෂණ

1. සරල සහ භාවිතයට පහසු (Simple)

ජාවා භාෂාව ඉගෙන ගැනීම සහ භාවිතා කිරීම පහසුය. එය C++ වැනි භාෂාවල සංකීර්ණ ලක්ෂණ (උදා: පොයින්ටර්) ඉවත් කර ඇත.

2. ඔබ්ජෙක්ට්-ඔරියන්ටඩ් (Object-Oriented)

ජාවා භාෂාව සම්පූර්ණයෙන්ම ඔබ්ජෙක්ට්-ඔරියන්ටඩ් වේ. එහි සියලුම දත්ත සහ ක්‍රියාකාරකම් ඔබ්ජෙක්ට් ආකාරයෙන් සංවිධානය වේ.

3. ප්ලැට්ෆෝම්-ස්වාධීන (Platform-Independent)

ජාවා කේතය "Write Once, Run Anywhere" (WORA) මූලධර්මය අනුගමනය කරයි. එයින් අදහස් කරන්නේ ජාවා කේතය ඕනෑම ප්ලැට්ෆෝමයක (Windows, Linux, Mac, ආදිය) ක්‍රියාත්මක කළ හැකි බවයි. මෙය හැකි වන්නේ ජාවා වර්චුවල මෂින් (Java Virtual Machine - JVM) නිසාය.

4. ආරක්ෂිත (Secure)

ජාවා භාෂාව ආරක්ෂිත ප්‍රෝග්‍රැමින් පරිසරයක් සපයයි. එහි ආරක්ෂිත ලක්ෂණ (උදා: බයිට්කේත සත්‍යාපනය, සන්ධිස්ථාන ආරක්ෂාව) නිසා එය අන්තර්ජාල යෙදුම් සඳහා බහුලව භාවිතා වේ.

5. බහු-නිධි (Multithreaded)

ජාවා භාෂාව බහු-නිධි (Multithreading) සහාය ලබා දෙයි. මෙයින් අදහස් කරන්නේ එක් යෙදුමක් තුළ එකවර කාර්යයන් බහුතරයක් ක්‍රියාත්මක කළ හැකි බවයි.

6. උසස් කාර්ය සාධනය (High Performance)

ජාවා කේතය බයිට්කේත (Bytecode) බවට පරිවර්තනය වන අතර, එය JVM මගින් ක්‍රියාත්මක වේ. මෙම ක්‍රියාවලිය ජාවා යෙදුම්වල කාර්ය සාධනය ඉහළ නංවයි.

ජාවාවේ වාසි

- ප්ලැට්ෆෝම්-ස්වාධීන භාෂාවකි.
- විශාල ප්‍රජා සහාය (Community Support)
- බහුලව භාවිතා වන පුස්තකාල (Libraries) සහ රාමු (Frameworks)
- ආරක්ෂිත සහ විශ්වසනීය
- බහු-නිධි සහාය (Multithreading)

ජාවා කේතයක උදාහරණය

පහත දැක්වෙන්නේ සරල "Hello, World!" ජාවා කේතයකි:


```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

- `public class HelloWorld` : මෙය ක්ලාස් එකක් නිර්වචනය කරයි. ජාවාවේ සියලුම කේතයන් ක්ලාස් තුළ ලියා ඇත.
- `public static void main(String[] args)` : මෙය ප්‍රධාන ක්‍රමය (Main Method) වේ. එය ජාවා ප්‍රෝග්‍රෑම් එකක් ආරම්භ කිරීමේ ලක්ෂ්‍යයයි.

ජාවා භාවිතා කරන යෙදුම්

- වෙබ් යෙදුම් (Web Applications)

ජාවා භාවිතා කරමින් වෙබ් යෙදුම් සංවර්ධනය කිරීම සඳහා Spring, Hibernate වැනි රාමු (Frameworks) භාවිතා කරයි.

- මොබයිල් යෙදුම් (Mobile Applications)

Android යෙදුම් සංවර්ධනය සඳහා ජාවා ප්‍රධාන භාෂාවක් ලෙස භාවිතා වේ.

- ඩෙස්ක්ටොප් යෙදුම් (Desktop Applications)

JavaFX සහ Swing වැනි ටූල්ස් භාවිතා කරමින් ඩෙස්ක්ටොප් යෙදුම් සංවර්ධනය කළ හැක.

- විශාල දත්ත සැකසුම් (Big Data Processing)

Hadoop, Apache Spark වැනි ටූල්ස් ජාවා භාවිතා කරමින් විශාල දත්ත සැකසුම් කරයි.

ජාවා සංවර්ධනය සඳහා අවශ්‍ය මෘලස්ථාන**(Environment)**

- JDK (Java Development Kit) : ජාවා කේතය සංවර්ධනය සඳහා අවශ්‍ය මෘලස්ථාන.

- IDE (Integrated Development Environment) : Eclipse, IntelliJ IDEA, NetBeans වැනි IDE භාවිතා කරමින් ජාවා කේතය ලිවීම සහ පරීක්ෂා කිරීම පහසු වේ.

ජාවා යනු බලගතු, බහුකාර්ය, සහ ඉහළ ඉල්ලුමක් ඇති ප්‍රෝග්‍රැමින් භාෂාවකි. එය ආරම්භකයින් සඳහා හොඳම භාෂාවක් ලෙස සැලකේ.

ක්ලාස් (Classes) සහ ඔබ්ජෙක්ට් (Objects)

ඔබ්ජෙක්ට්-ඔරියන්ටඩ් ප්‍රෝග්‍රැමින් (OOP) හි ක්ලාස් සහ ඔබ්ජෙක්ට් යනු මූලික සංකල්ප වේ. මෙම සංකල්ප භාවිතා කරමින් අපට යථාර්ථවාදී ලෝකයේ ඇති වස්තූන් (Objects) පරිගණක වැඩසටහන් තුළ නිරූපණය කළ හැක.

ක්ලාස් (Class)

ක්ලාස් යනු ඔබ්ජෙක්ට් සඳහා නිර්මාණ රටාවකි හෝ බ්ලූප්‍රින්ට් එකකි. එය ඔබ්ජෙක්ට්වල ගුණාංග (Attributes) සහ හැසිරීම් (Methods) නිර්වචනය කරයි. ක්ලාස් එකක් භාවිතා කරමින් ඔබට බහුලව ඔබ්ජෙක්ට් නිර්මාණය කළ හැක.

ක්ලාස් එකක උදාහරණය:

```
public class Car {
    // Attributes (ගුණාංග)
    String color;
    int speed;

    // Methods (හැසිරීම්)
    void drive() {
        System.out.println("The car is driving.");
    }

    void brake() {
        System.out.println("The car is braking.");
    }
}
```

මෙම උදාහරණයේ, `Car` ක්ලාස් එකේ `color` සහ `speed` යන ගුණාංග (Attributes) සහ `drive()` සහ `brake()` යන හැසිරීම් (Methods) ඇත.

ඔබ්ජෙක්ට් (Object)

ඔබ්ජෙක්ට් යනු ක්ලාස් එකක භෞතික නිදසුනකි (Instance). එය ක්ලාස් එකේ ගුණාංග සහ හැසිරීම් භාවිතා කරයි. ඔබ්ජෙක්ට් එකක් නිර්මාණය කිරීම සඳහා `new` යතුරුපදය භාවිතා කරයි.

ඔබ්ජෙක්ට් එකක් නිර්මාණය කිරීමේ උදාහරණය:

```
public class Main {
    public static void main(String[] args) {
        // Object creation (ඔබ්ජෙක්ට් නිර්මාණය)
        Car myCar = new Car();

        // Accessing attributes (ගුණාංග ප්‍රවේශ වීම)
        myCar.color = "Red";
        myCar.speed = 60;

        // Calling methods (හැසිරීම් ක්‍රියාත්මක කිරීම)
        myCar.drive();
        myCar.brake();
    }
}
```

මෙම උදාහරණයේ, `myCar` යනු `Car` ක්ලාස් එකේ ඔබ්ජෙක්ට් එකකි. එය `color` සහ `speed` ගුණාංග සහ `drive()` සහ `brake()` හැසිරීම් භාවිතා කරයි.

OOP

ක්ලාස් සහ ඔබ්ජෙක්ට් අතර වෙනස

- ක්ලාස් (Class) : ක්ලාස් යනු නිර්වචනයකි (Blueprint). එය ඔබ්ජෙක්ට්වල ගුණාංග සහ හැසිරීම් නිර්වචනය කරයි.

- ඔබ්ජෙක්ට් (Object) : ඔබ්ජෙක්ට් යනු ක්ලාස් එකක භෞතික නිදසුනකි (Instance). එය ක්ලාස් එකේ ගුණාංග සහ හැසිරීම් භාවිතා කරයි.

ක්ලාස් සහ ඔබ්ජෙක්ට් භාවිතා කිරීමේ පියවර

1. ක්ලාස් එකක් නිර්වචනය කරන්න (Define a Class)

ඔබට අවශ්‍ය ගුණාංග සහ හැසිරීම් සහිත ක්ලාස් එකක් නිර්වචනය කරන්න.

2. ඔබ්ජෙක්ට් එකක් නිර්මාණය කරන්න (Create an Object)

`new` යතුරුපදය භාවිතා කරමින් ක්ලාස් එකෙන් ඔබ්ජෙක්ට් එකක් නිර්මාණය කරන්න.

3. ගුණාංග ප්‍රවේශ වීම සහ හැසිරීම් ක්‍රියාත්මක කිරීම (Access Attributes and Methods)

ඔබ්ජෙක්ට් එක භාවිතා කරමින් ගුණාංග ප්‍රවේශ වීම සහ හැසිරීම් ක්‍රියාත්මක කිරීම.

උදාහරණය: ක්ලාස් සහ ඔබ්ජෙක්ට්

OOP

```
public class Dog {
    // Attributes (ගුණාංග)
    String breed;
    int age;

    // Methods (හැසිරීම්)
    void bark() {
        System.out.println("Woof! Woof!");
    }

    void sleep() {
        System.out.println("Zzz...");
    }
}

public class Main {
    public static void main(String[] args) {
        // Object creation (ඔබ්ජෙක්ට් නිර්මාණය)
        Dog myDog = new Dog();

        // Accessing attributes (ගුණාංග ප්‍රවේශ වීම)
        myDog.breed = "Golden Retriever";
        myDog.age = 3;

        // Calling methods (හැසිරීම් ක්‍රියාත්මක කිරීම)
        myDog.bark();
        myDog.sleep();
    }
}
```

ප්‍රතිදානය:

Woof! Woof!

Zzz...

OOP

සාරාංශය

- ක්ලාස් (Class) : ඔබ්ජෙක්ට් සඳහා නිර්මාණ රටාවකි.
- ඔබ්ජෙක්ට් (Object) : ක්ලාස් එකක භෞතික නිදසුනකි.
- ගුණාංග (Attributes) : ඔබ්ජෙක්ට්වල දත්ත (Data).
- හැසිරීම් (Methods) : ඔබ්ජෙක්ට්වල ක්‍රියාකාරකම් (Functions).

ක්ලාස් සහ ඔබ්ජෙක්ට් භාවිතා කිරීමෙන් ඔබට යථාර්ථවාදී ලෝකයේ වස්තූන් පරිගණක වැඩසටහන් තුළ නිරූපණය කළ හැකි අතර, කේතය වඩාත් සංවිධානාත්මක, නැවත භාවිතයට හැකි, සහ පාලනය කිරීමට පහසු වේ.

එන්කැප්සුලේෂන් (Encapsulation) සහ ඇබ්ස්ට්‍රැක්ෂන් (Abstraction)

ඔබ්ජෙක්ට්-ඔරියන්ටඩ් ප්‍රෝග්‍රැමින් (OOP) හි එන්කැප්සුලේෂන් සහ ඇබ්ස්ට්‍රැක්ෂන් යනු මූලික සංකල්ප දෙකකි. මෙම සංකල්ප භාවිතා කිරීමෙන් කේතය වඩාත් සුරක්ෂිත, සංවිධානාත්මක, සහ පාලනය කිරීමට පහසු වේ.

එන්කැප්සුලේෂන් (Encapsulation)

එන්කැප්සුලේෂන් යනු දත්ත සහ ක්‍රම (Methods) එක් ඒකකයක් තුළ ඇසුරුම් කිරීමේ ක්‍රමයයි. එය දත්ත සුරක්ෂිත කිරීම සහ කේතය පහසුවෙන් පාලනය කිරීමට උපකාරී වේ. එන්කැප්සුලේෂන් හි ප්‍රධාන අදහස නම්, දත්ත සෘජුව ප්‍රවේශ වීම වැළක්වීම සහ දත්ත ප්‍රවේශ වීම පාලනය කිරීමයි.

එන්කැප්සුලේෂන් හි ප්‍රධාන ලක්ෂණ:

1. දත්ත සඟවා ගැනීම (Data Hiding) : දත්ත සෘජුව ප්‍රවේශ වීම වැළක්වීම.
2. ප්‍රවේශ නියාමක (Access Modifiers) : `private`, `public`, `protected` යන ප්‍රවේශ නියාමක භාවිතා කරමින් දත්ත ප්‍රවේශ වීම පාලනය කිරීම.
3. Getters සහ Setters : දත්ත ප්‍රවේශ වීම සහ වෙනස් කිරීම සඳහා විශේෂ ක්‍රම භාවිතා කිරීම.

උදාහරණය:

```
public class Person {
    // Private attributes (දත්ත සඟවා ගැනීම)
    private String name;
    private int age;

    // Getter method for name
    public String getName() {
        return name;
    }

    // Setter method for name
    public void setName(String name) {
        this.name = name;
    }

    // Getter method for age
    public int getAge() {
        return age;
    }

    // Setter method for age
    public void setAge(int age) {
        if (age > 0) { // Validation (සත්‍යාපනය)
            this.age = age;
        }
    }
}

public class Main {
    public static void main(String[] args) {
        Person person = new Person();
        person.setName("John");
        person.setAge(25);

        System.out.println("Name: " + person.getName());
        System.out.println("Age: " + person.getAge());
    }
}
```

ප්‍රතිදානය:

Name: John

Age: 25

මෙම උදාහරණයේ, `name` සහ `age` යන දත්ත `private` ලෙස සඟවා ඇත. ඒවා ප්‍රවේශ වීම සඳහා `getter` සහ `setter` ක්‍රම භාවිතා කරයි.

ඇබ්ස්ට්‍රැක්ෂන් (Abstraction)

ඇබ්ස්ට්‍රැක්ෂන් යනු සංකීර්ණ තත්ත්වයන් සරල කිරීමේ ක්‍රමයකි. එය අවශ්‍ය තොරතුරු පමණක් පෙන්වන අතර, අනවශ්‍ය තොරතුරු සඟවයි. ඇබ්ස්ට්‍රැක්ෂන් භාවිතා කිරීමෙන් කේතය වඩාත් පැහැදිලි සහ පාලනය කිරීමට පහසු වේ.

ඇබ්ස්ට්‍රැක්ෂන් හි ප්‍රධාන ලක්ෂණ:

1. සංකීර්ණතාවය සඟවා ගැනීම (Hiding Complexity) : අවශ්‍ය නොවන තොරතුරු සඟවා ගැනීම.

2. අමුත්තන් (Interfaces) සහ ඇබ්ස්ට්‍රැක්ට් ක්ලාස් (Abstract Classes) : ඇබ්ස්ට්‍රැක්ෂන් ක්‍රියාත්මක කිරීම සඳහා අමුත්තන් සහ ඇබ්ස්ට්‍රැක්ට් ක්ලාස් භාවිතා කිරීම.

උදාහරණය:

```
// Abstract class (ඇබ්ස්ට්‍රැක්ට් ක්ලාස්)
abstract class Animal {
    // Abstract method (ඇබ්ස්ට්‍රැක්ට් ක්‍රමය)
    abstract void makeSound();

    // Concrete method (සාමාන්‍ය ක්‍රමය)
    void sleep() {
        System.out.println("Zzz...");
    }
}

// Concrete class (සාමාන්‍ය ක්ලාස්)
class Dog extends Animal {
    @Override
    void makeSound() {
        System.out.println("Woof! Woof!");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.makeSound(); // Abstract method
        dog.sleep();     // Concrete method
    }
}
```

OOP

ප්‍රතිදානය:

Woof! Woof!

Zzz...

මෙම උදාහරණයේ, `Animal` ඇබ්ස්ට්‍රැක්ට් ක්ලාස් එක `makeSound()` ඇබ්ස්ට්‍රැක්ට් ක්‍රමයක් සහ `sleep()` සාමාන්‍ය ක්‍රමයක් අර්ථ දක්වයි. `Dog` ක්ලාස් එක `Animal` ක්ලාස් එකෙන් උරුම වී `makeSound()` ක්‍රමය ක්‍රියාත්මක කරයි.

සාරාංශය

- එන්කැප්සුලේෂන් (Encapsulation) : දත්ත සහ ක්‍රම එක් ඒකකයක් තුළ ඇසුරුම් කිරීම. දත්ත සුරක්ෂිත කිරීම සහ ප්‍රවේශ නියාමනය.

- ඇබ්ස්ට්‍රැක්ෂන් (Abstraction) : සංකීර්ණතාවය සඟවා ගැනීම සහ අවශ්‍ය තොරතුරු පමණක් පෙන්වීම. කේතය සරල කිරීම සහ පාලනය කිරීම.

මෙම සංකල්ප දෙක භාවිතා කිරීමෙන් ඔබේ කේතය වඩාත් සුරක්ෂිත, සංවිධානාත්මක, සහ පාලනය කිරීමට පහසු වේ.

මෙතෝඩස් (Methods)

මෙතෝඩස් (Methods) යනු ඔබ්ජෙක්ට්-ඔරියන්ටඩ් ප්‍රෝග්‍රැමින් (OOP) හි මූලික සංකල්පයකි. මෙතෝඩස් භාවිතා කරමින් අපට කේතය නැවත භාවිතා කිරීම, සංවිධානය කිරීම, සහ පාලනය කිරීම පහසු වේ. මෙතෝඩස් යනු ක්ලාස් එකක් තුළ අර්ථ දක්වා ඇති ක්‍රියාකාරකම් (Functions) වේ.

මෙතෝඩස් හි ව්‍යුහය

මෙතෝඩ් එකක ව්‍යුහය පහත පරිදි වේ:

```
accessModifier returnType methodName(parameterList) {
    // Method body (ක්‍රමයේ ශරීරය)
}
```

- `accessModifier` : මෙතෝඩ් එකට ප්‍රවේශ වීමේ මට්ටම (උදා: `public`, `private`, `protected`).
- `returnType` : මෙතෝඩ් එක ආපසු ලබා දෙන දත්ත වර්ගය (උදා: `int`, `String`, `void`).
- `methodName` : මෙතෝඩ් එකේ නම.
- `parameterList` : මෙතෝඩ් එකට ලබා දෙන පරාමිතීන් (Parameters).

මෙතෝඩස් වර්ග

1. පරාමිති රහිත මෙතෝඩස් (Methods without Parameters)

   මෙතෝඩස් පරාමිති ලබා නොගනී.

උදාහරණය:

```
public void greet() {
    System.out.println("Hello, World!");
}
```

2. පරාමිති සහිත මෙතෝඩස් (Methods with Parameters)

මෙතෝඩස් පරාමිති ලබා ගනී.

උදාහරණය:

```
public void greet(String name) {
    System.out.println("Hello, " + name + "!");
}
```

3. ආපසු අගය ලබා දෙන මෙතෝඩස් (Methods with Return Values)

මෙතෝඩස් ආපසු අගයක් ලබා දෙයි.

උදාහරණය:


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

4. ආපසු අගය නොලබා දෙන මෙතෝඩස් (Void Methods)

මෙතෝඩස් ආපසු අගයක් නොලබා දෙයි.

උදාහරණය:

```
public void printMessage(String message) {
    System.out.println(message);
}
```

මෙතෝඩස් භාවිතා කිරීමේ උදාහරණ

```
public class Calculator {
   // Method to add two numbers
   public int add(int a, int b) {
      return a + b;
   }

   // Method to subtract two numbers
   public int subtract(int a, int b) {
      return a - b;
   }

   // Method to multiply two numbers
   public int multiply(int a, int b) {
      return a * b;
   }

   // Method to divide two numbers
   public double divide(int a, int b) {
      if (b != 0) {
         return (double) a / b;
      } else {
         System.out.println("Division by zero is not allowed.");
         return 0;
      }
   }
}

public class Main {
   public static void main(String[] args) {
      Calculator calc = new Calculator();

      int sum = calc.add(10, 5);
      int difference = calc.subtract(10, 5);
      int product = calc.multiply(10, 5);
      double quotient = calc.divide(10, 5);

      System.out.println("Sum: " + sum);
      System.out.println("Difference: " + difference);
      System.out.println("Product: " + product);
      System.out.println("Quotient: " + quotient);
   }
}
```

ප්‍රතිදානය:

Sum: 15

Difference: 5

Product: 50

Quotient: 2.0

මෙතෝඩ් ඕවර්ලෝඩින් (Method Overloading)

මෙතෝඩ් ඕවර්ලෝඩින් යනු එකම නමක් ඇති මෙතෝඩස් බහුලව නිර්වචනය කිරීමේ ක්‍රමයයි. මෙය පරාමිති ලැයිස්තුව (Parameter List) වෙනස් කිරීමෙන් ක්‍රියාත්මක වේ.

OOP

උදාහරණය:

```
public class MathOperations {
   // Method to add two integers
   public int add(int a, int b) {
      return a + b;
   }

   // Method to add three integers
   public int add(int a, int b, int c) {
      return a + b + c;
   }

   // Method to add two double values
   public double add(double a, double b) {
      return a + b;
   }
}

public class Main {
   public static void main(String[] args) {
      MathOperations math = new MathOperations();

      System.out.println("Sum of 10 and 5: " + math.add(10, 5));
      System.out.println("Sum of 10, 5, and 15: " + math.add(10, 5, 15));
      System.out.println("Sum of 10.5 and 5.5: " + math.add(10.5, 5.5));
   }
}
```

ප්‍රතිදානය:

Sum of 10 and 5: 15

Sum of 10, 5, and 15: 30

Sum of 10.5 and 5.5: 16.0

OOP

සාරාංශය

- මෙතෝඩස් (Methods) : ක්ලාස් එකක් තුළ අර්ථ දක්වා ඇති ක්‍රියාකාරකම්.

- පරාමිති (Parameters) : මෙතෝඩස් ලබා ගන්නා අගයන්.

- ආපසු අගය (Return Value) : මෙතෝඩස් ආපසු ලබා දෙන අගය.

- මෙතෝඩ් ඕවර්ලෝඩින් (Method Overloading) : එකම නමක් ඇති මෙතෝඩස් බහුලව නිර්වචනය කිරීම.

මෙතෝඩස් භාවිතා කිරීමෙන් ඔබේ කේතය වඩාත් සංවිධානාත්මක, නැවත භාවිතයට හැකි, සහ පාලනය කිරීමට පහසු වේ.

Control Statements (පාලන ප්‍රකාශන)

Control Statements යනු ප්‍රෝග්‍රෑම් එකක ක්‍රියාකාරිත්වය පාලනය කිරීම සඳහා භාවිතා වන ප්‍රකාශන වේ. මෙම ප්‍රකාශන භාවිතා කරමින් ඔබට කේතයේ ධාරාව (Flow) පාලනය කිරීම, කොන්දේසි පරීක්ෂා කිරීම, සහ පුනරාවර්තන (Loops) ක්‍රියාත්මක කිරීම පහසු වේ.

Control Statements වර්ග

1. **Decision-Making Statements (තීරණ ගැනීමේ ප්‍රකාශන)**
   - `if`
   - `if-else`
   - `if-else-if`
   - `switch`

2. **Looping Statements (පුනරාවර්තන ප්‍රකාශන)**
   - `for`
   - `while`
   - `do-while`

3. **Jump Statements (පනින්න ප්‍රකාශන)**
   - `break`
   - `continue`
   - `return`

1. Decision-Making Statements (තීරණ ගැනීමේ ප්‍රකාශන)

`if` Statement

`if` ප්‍රකාශනය භාවිතා කරමින් කොන්දේසියක් පරීක්ෂා කරයි. කොන්දේසිය සත්‍ය (True) නම්, `if` බ්ලොක් එක තුළ ඇති කේතය ක්‍රියාත්මක වේ.

උදාහරණය:

```
int age = 18;
if (age >= 18) {
    System.out.println("You are eligible to vote.");
}
```

ප්‍රතිදානය:

You are eligible to vote.

`if-else` Statement

`if-else` ප්‍රකාශනය භාවිතා කරමින් කොන්දේසියක් පරීක්ෂා කරයි. කොන්දේසිය සත්‍ය (True) නම්, `if` බ්ලොක් එක තුළ ඇති කේතය ක්‍රියාත්මක වේ. නැතහොත්, `else` බ්ලොක් එක තුළ ඇති කේතය ක්‍රියාත්මක වේ.

උදාහරණය:

```
int age = 16;
if (age >= 18) {
    System.out.println("You are eligible to vote.");
} else {
    System.out.println("You are not eligible to vote.");
}
```

ප්‍රතිදානය:

You are not eligible to vote.

`if-else-if` Statement

`if-else-if` ප්‍රකාශනය භාවිතා කරමින් බහු කොන්දේසි පරීක්ෂා කරයි.

උදාහරණය:

```
int marks = 85;
if (marks >= 90) {
    System.out.println("Grade: A");
} else if (marks >= 80) {
    System.out.println("Grade: B");
} else if (marks >= 70) {
    System.out.println("Grade: C");
} else {
    System.out.println("Grade: D");
}
```

ප්‍රතිදානය:

Grade: B

`switch` Statement

`switch` ප්‍රකාශනය භාවිතා කරමින් බහු කොන්දේසි පරීක්ෂා කරයි. එය `case` භාවිතා කරමින් විවිධ කොන්දේසි සඳහා කේතය ක්‍රියාත්මක කරයි.

උදාහරණය:

```
int day = 3;
switch (day) {
   case 1:
      System.out.println("Monday");
      break;
   case 2:
      System.out.println("Tuesday");
      break;
   case 3:
      System.out.println("Wednesday");
      break;
   default:
      System.out.println("Invalid day");
}
```

ප්‍රතිදානය:

Wednesday

2. Looping Statements (පුනරාවර්තන ප්‍රකාශන)

`for` Loop

`for` පුනරාවර්තනය භාවිතා කරමින් කේතය නිශ්චිත වාර ගණනක් ක්‍රියාත්මක කරයි.

උදාහරණය:

```
for (int i = 1; i <= 5; i++) {
    System.out.println("Iteration: " + i);
}
```

ප්‍රතිදානය:

Iteration: 1

Iteration: 2

Iteration: 3

Iteration: 4

Iteration: 5

`while` Loop

`while` පුනරාවර්තනය භාවිතා කරමින් කොන්දේසියක් සත්‍ය (True) වන තෙක් කේතය ක්‍රියාත්මක කරයි.

උදාහරණය:

```
int i = 1;
while (i <= 5) {
    System.out.println("Iteration: " + i);
    i++;
}
```

ප්‍රතිදානය:

Iteration: 1

Iteration: 2

Iteration: 3

Iteration: 4

Iteration: 5

`do-while` Loop

`do-while` පුනරාවර්තනය භාවිතා කරමින් කේතය අවම වශයෙන් එක් වරක් ක්‍රියාත්මක කරයි. පසුව, කොන්දේසිය සත්‍ය (True) වන තෙක් කේතය ක්‍රියාත්මක වේ.

උදාහරණය:

```
int i = 1;
do {
   System.out.println("Iteration: " + i);
   i++;
} while (i <= 5);
```

ප්‍රතිදානය:

Iteration: 1

Iteration: 2

Iteration: 3

Iteration: 4

Iteration: 5

3. Jump Statements (පනින්න ප්‍රකාශන)

`break` Statement

`break` ප්‍රකාශනය භාවිතා කරමින් පුනරාවර්තනයක් හෝ `switch` ප්‍රකාශනයක් තුළින් පිටවීම.

උදාහරණය:

```
for (int i = 1; i <= 10; i++) {
    if (i == 5) {
        break;
    }
    System.out.println("Iteration: " + i);
}
```


ප්‍රතිදානය:

Iteration: 1

Iteration: 2

Iteration: 3

Iteration: 4

`continue` Statement

`continue` ප්‍රකාශනය භාවිතා කරමින් පුනරාවර්තනයක් තුළ ඉතිරි කේතය මඟ හරිමින් ඊළඟ පුනරාවර්තනයට යාම.

උදාහරණය:

```
for (int i = 1; i <= 5; i++) {
    if (i == 3) {
        continue;
    }
    System.out.println("Iteration: " + i);
}
```

ප්‍රතිදානය:

Iteration: 1

Iteration: 2

Iteration: 4

Iteration: 5

`return` Statement

`return` ප්‍රකාශනය භාවිතා කරමින් මෙතෝඩ් එකක් තුළින් ආපසු අගයක් ලබා දීම.

උදාහරණය:

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

Control Statements භාවිතා කිරීමෙන් ඔබේ ප්‍රෝග්‍රෑම් එකේ ධාරාව (Flow) පාලනය කිරීම සහ කේතය වඩාත් සංවිධානාත්මක කිරීම පහසු වේ.