CJ Unit 2

Control Flow Statement -

If-else Statement -

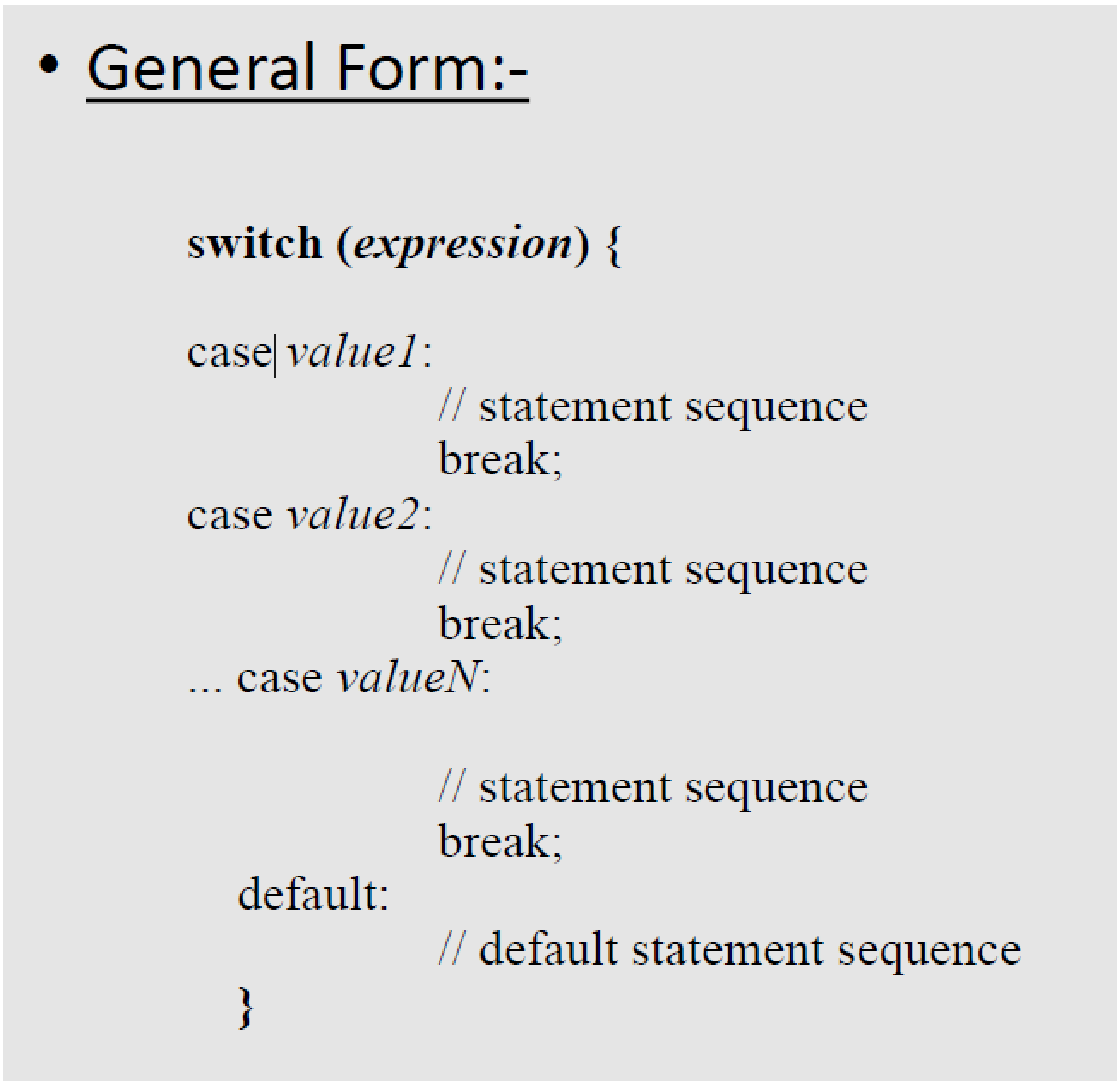
```
// Basic Syntaxs

if(condition){
//code to be executed
}

if(condition){
//code if condition is true
}else{
//code if condition is false
}

if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```

Switch Statement -



- General Form:-

```
switch (expression) {

case value1:
                // statement sequence
                break;
case value2:
                // statement sequence
                break;
... case valueN:

                // statement sequence
                break;
   default:
                // default statement sequence
   }
```

Iterations -

1. **while**

- It loops through a block of code as long as a specified condition is true.
- If the number of iterations is not fixed, it is recommended to use while loop.
- General Form:

```
while(condition)
// body of loop
}
```

2. **do while**

- This loop always executes its body at least once, because its conditional expression is at the bottom of the loop.
- General Form:
  ```
  do {
  // body of loop
  } while (condition)
  ```

3. **for**

- For loop is used When you know exactly how many times you want to loop through a block of code.
- General Form:
  ```
  for(initialization ; condition ; iteration){
  // body of the loop
  }
  ```

4. **for each**

- The for each loop allows iterating over arrays and other collections in sequential fashion from start to finish.
- General Form:
  ```
  for(data_type variable array collection){
  // body of the loop
  }
  ```

Jump Statements -

1. **break -**

- In java, the break statement is used to terminate the execution of the nearest looping statement or switch statement.
- The break statement is widely used with the switch statement, **for** loop, **while** loop, **do-while** loop.
- Syntax - break;

2. **continue -**

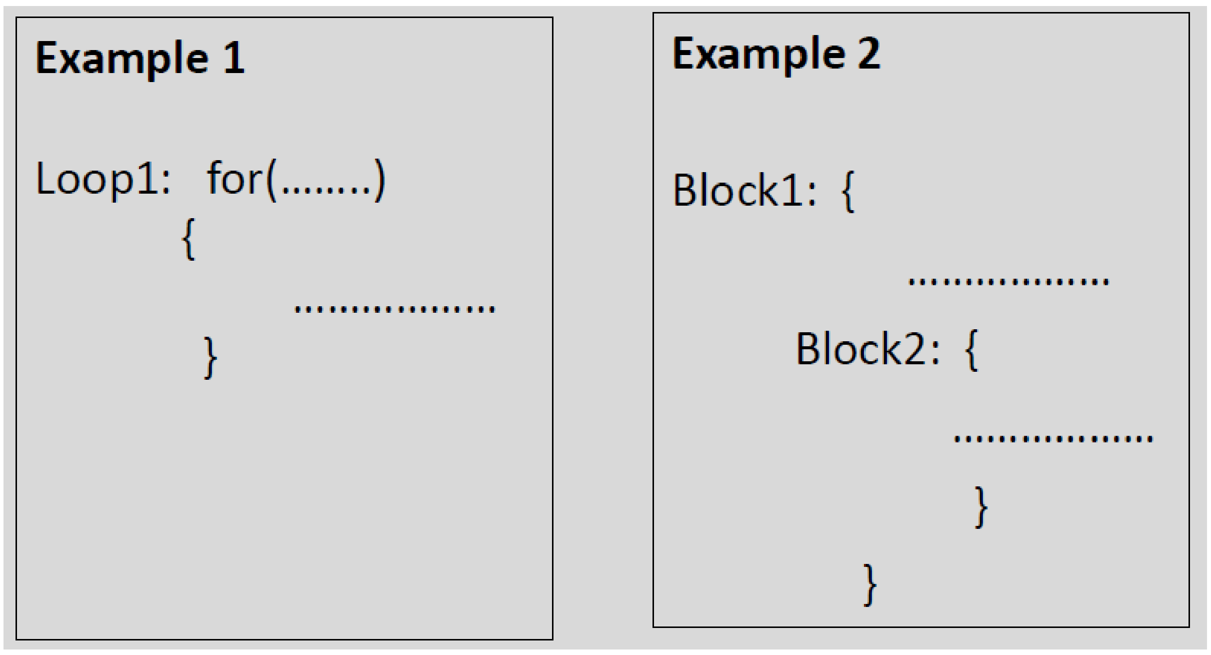
- The **continue** statement pushes the next repetition of the loop to take place, hopping any code between itself and the conditional expression that controls the loop.
- Syntax - continue;

3. **return -**

- *Then return statement terminates the execution of the current method and passes the control to the calling method.*
- Syntax - return;

Labelled Loops -

- In java, we can give a label to a block of statements.
- A label is any valid java variable name.



Example 1

```
Loop1:   for(........)
         {
               ..................
          }
```

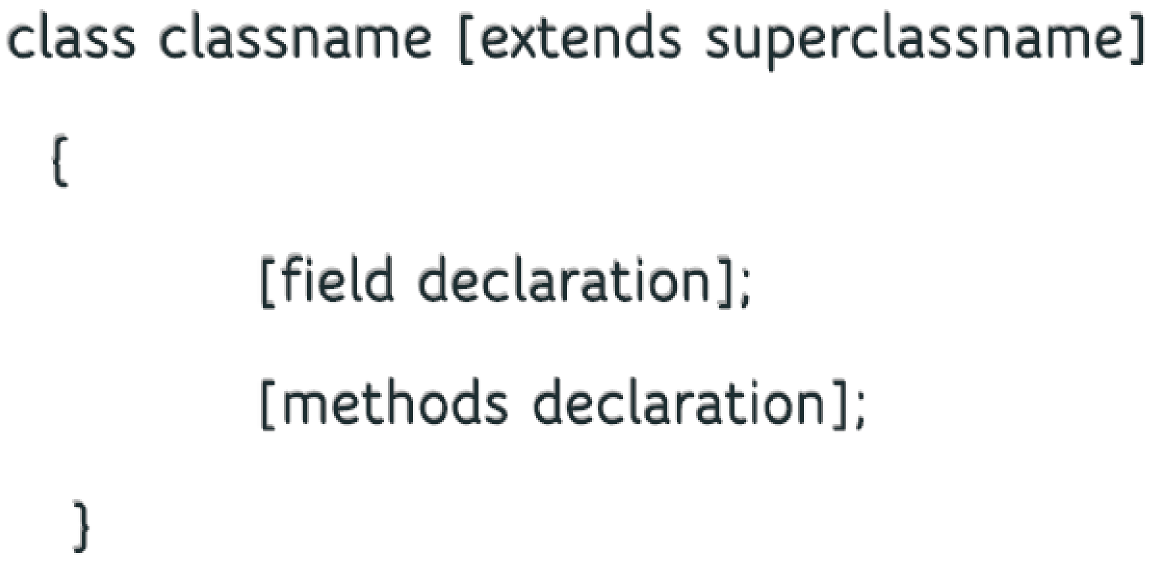
Example 2

```
Block1: {
              ..................
        Block2: {
                   ..................
                 }
         }
```

Class -

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity.

General Structure -



```
class classname [extends superclassname]
 {
           [field declaration];
           [methods declaration];
  }
```

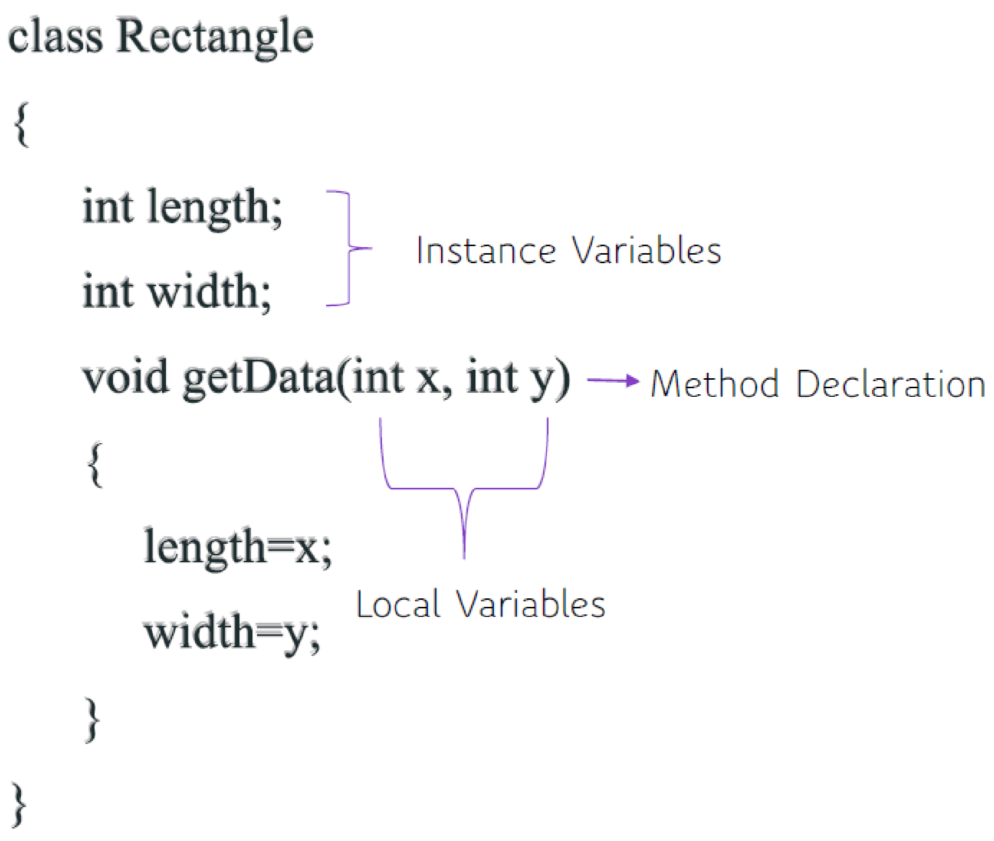
Field Declarations -

- The data, or variables, defined within a class are called instance variables.
- The code is contained within methods.
- Collectively, the methods and variables defined within a class are called members of the class.
- As a general rule, it is the methods that determine how a class' data can be used.

Method -

- Methods are declared inside the body of the class but immediately after the declaration of instance variables.
- General Form:

```
type method_name(parameter-list)
{
    Method - body;
}
```



```
class Rectangle
{
    int length;
    int width;
    void getData(int x, int y)
    {
        length=x;
        width=y;
    }
}
```

Creating Object -

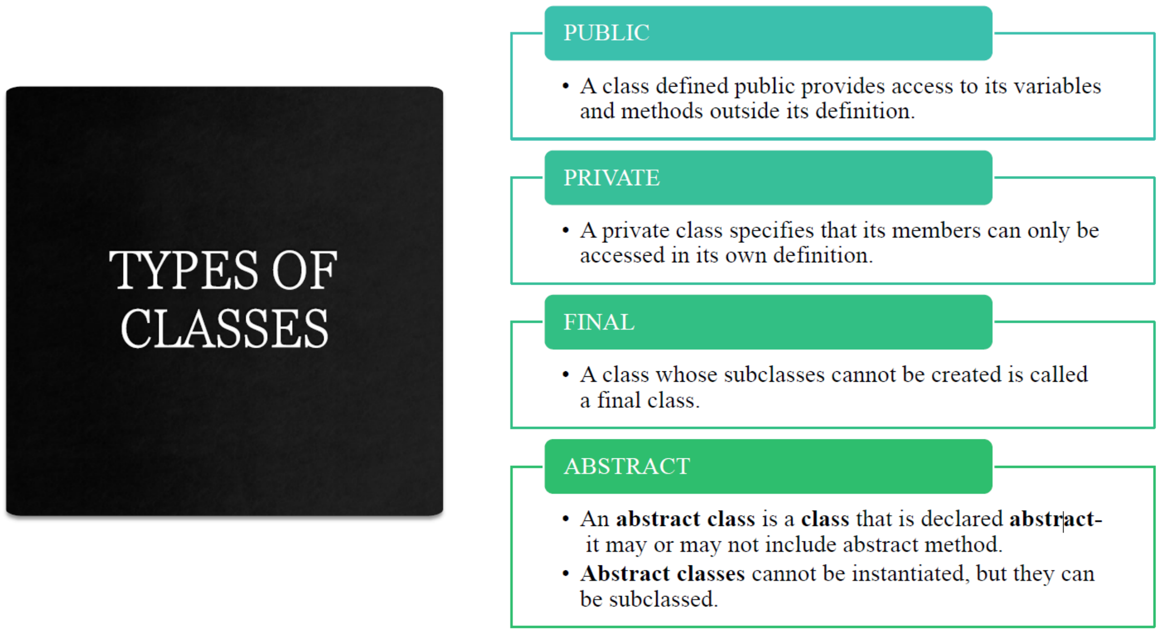
- Objects in java are created using the new operator.

- The new operator creates an object of the specified class and returns a reference to that object.
- **Example -**

Rectangle rect1; rect1;//declare the object rect1=new

Rectangle(); //instantiate the object

- Both statements can be combined
- Rectangle rect1=new Rectangle();



TYPES OF CLASSES

PUBLIC

- A class defined public provides access to its variables and methods outside its definition.

PRIVATE

- A private class specifies that its members can only be accessed in its own definition.

FINAL

- A class whose subclasses cannot be created is called a final class.

ABSTRACT

- An **abstract class** is a **class** that is declared **abstract**- it may or may not include abstract method.
- **Abstract classes** cannot be instantiated, but they can be subclassed.

Method Overloading -

- If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.
- Concept of Polymorphism.
- There are two ways to overload the method in java -

1. By changing number of arguments.
2. By changing the data type.

```
// Changing number of Arguments
```

```
class Adder{
int add(int
a,int b){
return
a+b
}
int add(int
a,int b,int c)
{
return
a+b+c
}
```

```
// Changing the Data Type

class Adder{
int add(int a, int b)
{
return
a+b
}
double add(double a, double b)
{
return
a+b
}
}
```

Variable Arguments -

- variable number of parameters for a given argument. Vararg must be the last argument in the
  formal argument list.

```
// SYNTAX

return_type method_name(data_type… variableName){
    // body of method
}

// Ex.
public void show(String… records){ } // definition
show(“VSIT”,”VP”,”VIT”); // Method Calling
```

Constructors -

- Java constructors or constructors in Java is a terminology used to construct something in our programs.
- A constructor in Java is a **special method** that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes.
- In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created.
- At the time of calling the constructor, memory for the object is allocated in the memory. It is a special type of method which is used to initialize the object.
- Every time an object is created using the new() keyword, at least one constructor is called.

Types of Constructors -

1. **DEFAULT** - When it doesn't have any parameter.
2. **PARAMETERIZED** - Having a specific number of parameters is called a parameterized constructor.
3. **COPY** - creates an object using another object of the same Java class.

Constructor Overloading -

The constructor overloading can be defined as the concept of having more than one constructor with different parameters so that every constructor can perform a different task.

```
public class Student {
//instance variables of the class
int id;
String name;

Student(){
System.out.println("this a default constructor");
}

Student(int i, String n){
id = i;
name = n;
}
public static void main(String[] args) {
//object creation
Student s = new Student();
System.out.println("\nDefault Constructor values: \n");
System.out.println("Student Id : "+s.id + "\nStudent Name : "+s.name);

System.out.println("\nParameterized Constructor values: \n");
Student student = new Student(10, "David");
System.out.println("Student Id : "+student.id + "\nStudent Name : "+student.name);
}
}

// Output

this a default constructor


Default Constructor values:


Student Id : 0
Student Name : null


Parameterized Constructor values:


Student Id : 10
Student Name : David
```

Static Variable -

- The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees, college name of students etc.
- The static variable gets memory only once in class area at the time of class loading.
- It is also known as Class Variable.
- Example :- static int a=20;

Static Method -

- A static method belongs to the class rather than object of a class.

- A static method can be invoked without the need for creating an instance of a class.
- Static method can access only static data member and can change the value of it.
- Example:
  static void getData(){} getData(){}// method definition
  getData() // calling of method

Nesting of Methods -

A method can be called by using only its name by another method of the same class. This is known as nesting of methods.

```
class Nesting{
int
m,n
int largest(){
if(m>=n)
return m;
else
return n;
}
void display(){
int large=largest();
largest();//calling a method
System.out.println
("Largest value="+
}
```

Garbage Collection -

- When no references to an object exist, that object is assumed to be no longer needed, and the memory occupied by the object can be reclaimed. There is no explicit need to destroy objects as in C++.
- Garbage collection only occurs sporadically (if at all) during the execution of your program. It will not occur simply because one or more objects exist that are no longer used.