Classes and Objects

Agenda

1. Classes & Objects
2. Static Block

Objectives

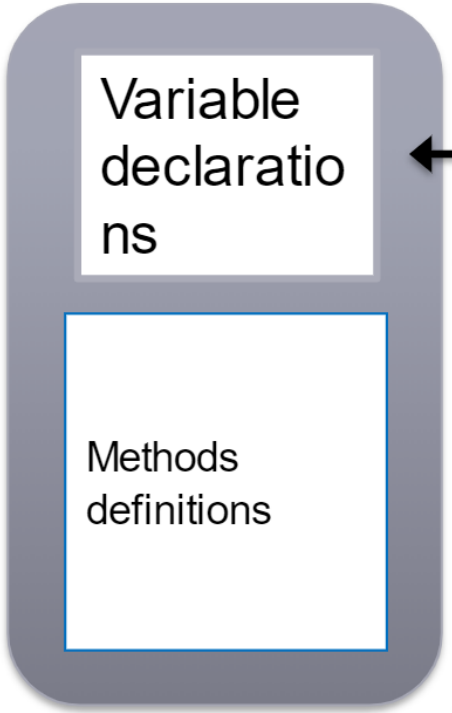
At the end of this module, you will be able to:

- Create classes and Objects
- Understand the importance of static block

Classes & Objects

Classes

A class contains variable declarations and method definitions



Variable declarations (variable describes the attributes)

Variable may be: instance variables or static variables or final variables

Method definitions (methods handle the behavior)

Methods may be: instance methods or static methods

Defining a Class in java

Define an Employee class with instance variables and instance methods

class Employee

Instance variables

- id
- name
- salary

Instance methods

- setId(…)
- setName(…)
- setSalary(…)
- getEmployeeDetails()

```
class Employee{
  int id;
  String name;
  int salary;

  void setId(int i) {
     id = i;
  }
  void setName(String n) {
     name = n;
  }
  void setSalary(int s) {
     salary = s;
  }
  void getEmployeeDetails( ) {
     System.out.println (name + " salary
is " + salary);
  }
}
```

Basic information about a class

```
public class Account {
   double balance;
   public void deposit( double amount ){
      balance += amount;
   }
   public double withdraw( double amount ){
      int minimum_balance=5000;
      if (balance >= (amount+minimum_balance)){
           balance -= amount;
           return amount;
      }
      else {
           System.out.println("Insufficient Balance");
           return 0.0;
      }    }
   public double getbalance(){
                return balance;
   }  }
```

Instance Variable

Parameter or argument

local Variable

Basic information about a class (Contd.).

```
    else {
        System.out.println("Insufficient Balance");
        return 0.0;
    }
  }
   public double getbalance(){
                return balance;
  }
}
```

Member variables

- The previous slide contains definition of a class called Accounts.
- A class contains members which can either be variables(fields) or methods(behaviors).
- A variable declared within a class(outside any method) is known as an **instance variable**.
- A variable declared within a method is known as **local variable**.
- Variables with method declarations are known as **parameters or arguments**.
- A class variable can also be declared as static where as a local variable cannot be static.

Objects and References

- Once a class is defined, you can declare a variable (object reference) of type class

```
Student stud1;
Employee emp1;
```

- The new operator is used to create an object of that reference type

```
Employee emp = new Employee();
```

Object reference

object

- Object references are used to store objects.
- Reference can be created for any type of classes (like concrete classes, abstract classes) and interfaces.

Objects and References (Contd.).

- The new operator,

  Dynamically allocates memory for an object

  Creates the object on the heap

  Returns a reference to it

  The reference is then stored in the variable

Employee class - Example

```
class Employee{
int id;
String name;
int salary;
void setId(int no){
id = no;
}
void setName(String n){
name = n;
}
void setSalary(int s){
salary = s;
}
void getEmployeeDetails(){
System.out.println(name + " salary is "+ salary);
}
}
public class EmployeeDemo {
public static void main(String[] args) {
Employee emp1 = new Employee();
emp1.setId(101);
emp1.setName("John");
emp1.setSalary(12000);
emp1.getEmployeeDetails();
}
}
```

Output:

John salary is 12000

Constructors

- While designing a class, the class designer can define within the class, a special method called 'constructor'
- Constructor is automatically invoked whenever an object of the class is created
- Rules to define a constructor
  - A constructor has the same name as the class name
  - A constructor should not have a return type
  - A constructor can be defined with any access specifier (like private, public)
  - A class can contain more than one constructor, So it can be overloaded

Constructor - Example

```
class Sample{
private int id;
Sample(){
id = 101;
System.out.println("Default constructor, with ID: "+id);
}
Sample(int no){
id = no;
System.out.println("One argument constructor,with ID: "+ id);
}
}
public class ConstDemo {
public static void main(String[] args) {
Sample s1 = new Sample();
Sample s2 = new Sample(102);
}
}
```

Output:
Default constructor, with ID: 101
One argument constructor,with ID: 102

this reference keyword

- Each class member function contains an implicit reference of its class type, named this
- this reference is created automatically by the compiler
- It contains the address of the object through which the function is invoked
- Use of this keyword
  - this can be used to refer instance variables when there is a clash with local variables or method arguments
  - this can be used to call overloaded constructors from another constructor of the same class

this Reference (Contd.).

- **Ex1:**

```
void  setId (int id){
      this.id = id;
}
```

instance variable

argument variable

- **Ex2:**

```
class Sample{
Sample(){
this("Java"); // calls overloaded constructor
System.out.println("Default constructor ");
}
Sample(String str){
System.out.println("One argument constructor "+ str);
}
}
```

this Reference (Contd.).

- Use this.variableName to explicitly refer to the instance variable.
- Use variable Name to refer to the parameter.
- The **this** reference is implicitly used to refer to instance variables and methods.
- It **CANNOT** be used in a static method.

Static Class Members

- Static class members are the members of a class that do not belong to an instance of a class
- We can access static members directly by prefixing the members with the class name

  ClassName.staticVariable

  ClassName.staticMethod(...)

Static variables:

- Shared among all objects of the class
- Only one copy exists for the entire class to use

Static Class Members (Contd.).

- Stored within the class code, separately from instance variables that describe an individual object
- Public static final variables are global constants

Static methods:

- Static methods can only access directly the static members and manipulate a class's static variables
- Static methods cannot access non-static members(instance variables or instance methods) of the class
- Static method cant access this and super references

Static Class Members – Example

```
class StaticDemo
{
private static int a = 0;
private int b;
public void set ( int i, int j)
{
a = i; b = j;
}
public void show( )
{
System.out.println("This is static a: " + a );
System.out.println( "This is non-static b: " + b );
}
```