# Core Java

## Day 04 Agenda

- Packages
- Access modifiers
- Arrays
  - 1-D array

## Class and Object

### "this" reference

- "this" is implicit reference variable that is available in every non-static method of class which is used to store reference of current/calling instance.
- Whenever any non-static method is called on any object, that object is internally passed to the method and internally collected in implicit "this" parameter.
- "this" is constant within method i.e. it cannot be assigned to another object or null within the method.
- Using "this" inside method (to access members) is optional. However, it is good practice for readability.
- In a few cases using "this" is necessary.
  - Unhide non-static fields from local fields.

    ```java
    double height; // "height" field
    void setHeight(double height) { // "height" parameter
        height = height; // ???
    }
    ```

  - Constructor chaining
  - Access instance of outer object.

### null reference

- In Java, local variables must be initialized before use; otherwise it raise compiler error.

    ```java
    int a;
    a++; // compiler error
    Human obj;
    obj.walk(); // compiler error
    ```

- In Java, reference can be initialized to null (if any specific object is not yet available). However reference must be initialized to appropriate object before its use; otherwise it will raise NullPointerException at runtime.

```
Human h = null; // reference initialized to null
h = new Human(); // reference initialized to the object
h.walk(); // invoke the method on the object
```

```
Human h = null;
h.walk(); // NullPointerException
```

## Constructor chaining

- Constructor chaining is executing a constructor of the class from another constructor (of the same class).

```
Human(int age, double height, double weight) {
    this.age = age;
    this.height = height;
    this.weight = weight;
}
Human() {
    this(0, 1.6, 3.3);
    // ...
}
```

- Constructor chaining (if done) must be on the very first line of the constructor.

# OOP concepts

## Abstraction

- Abstraction is getting essential details of the system.
- Abstraction change as per perspective of the user.
- Abstraction represents outer view of the system.
- Abstraction is based on interface/public methods of the class.

## Encapsulation

- Combining information/state and complex logic/operations so that it will be easier to use is called as Encapsulation.
- Fields and methods are bound in a class so that user can create object and invoke methods (without looking into complex implementations).
- Encapsulation represents inner view of the system.
- Encapsulation and abstraction are complementary to each other.

## Information hiding

- Members of class not intended to be visible outside the class can be restricted using access modifiers/specifiers.
- The "private" members can be accessed only within the class; while "public" members are accessible in as well as outside the class.
- Usually fields (data) are "private" and methods (operations) are "public".

# Packages

- Packages makes Java code modular. It does better organization of the code.

- Package is a container that is used to group logically related classes, interfaces, enums, and other packages.

- Package helps developer:

    - To group functionally related types together.
    - To avoid naming clashing/collision/conflict/ambiguity in source code.
    - To control the access to types.
    - To make easier to lookup classes in Java source code/docs.

- Java has many built-in packages.

    - java.lang * --> Integer, System, String, ...
    - java.util --> Scanner, ArrayList, Date, ...
    - java.io --> FileInputStream, FileOutputStream, ...
    - java.sql --> Connection, Statement, ResultSet, ...
    - java.util.function --> Predicate, Consumer, ...
    - javax.servlet.http --> HttpServlet, ...

- Package Syntax

    - To define a type inside package, it is mandatory write package declaration statement inside .java file.
    - Package declaration statement must be first statement in .java file.
    - Types inside package called as packaged types; while others (in default package) are unpackaged types.
    - Any type can be member of single package only.

- It is standard practice to have multi-level packages (instead of single level). Typically package name is module name, dept/project name, website name in reverse order.

```
package com.sunbeaminfo.modular.corejava;
```

```
package com.sunbeaminfo.dac;
```

- Packages can be used to avoid name clashing. In other words, two packages can have classes/types with same name.

```
package com.sunbeam.tree;

class Node {
    // ...
}

public class Tree {
    // ...
}
```

```
package com.sunbeam.list;

class Node {
    // ...
}

public class List {
    // ...
}
```

- When compiled, packages are created in form of directories (and sub-directories).

```
terminal> javac -d ..\bin FileName.java
```

- When setting CLASSPATH, one must give the path of the package (not the class).

```
# assumming the top-level package "com" is kept in "bin" directory.
terminal> set CLASSPATH=..\bin

# assuming the main() method is in Program class.
terminal> java com.sunbeaminfo.dac.Program
```

- Alternatively classpath, can be set with "java" command line argument.

```
# assuming the main() method is in Program class.
terminal> java -cp ../bin com.sunbeaminfo.dac.Program
```

- To use a class/type from another package, it should be "public" and must be imported using "import" keyword or use fully qualified name of the class/type. The CLASSPATH must be set before compiling the (dependent) class.

```java
import com.sunbeaminfo.tree.Tree;

class Program {
    public static void main(String[] args) {
        Tree t = new Tree();
        // ...
        com.sunbeaminfo.list.List l = new com.sunbeaminfo.list.List();
        // ...
    }
}
```

- To import all types in package use '*';

```java
import java.util.*;
```

- Importing static members of a class can be done using "import static" statement.

- Note: A class in package can never use/call a class without package.

- In eclipse, packages are mapped to directories under "src" directory.

## Access modifiers (for types)

- default: When no specifier is mentioned.
  - The types are accessible in current package only.
- public: When "public" specifier is mentioned.
  - The types are accessible in current package as well as outside the package (using import).

## Access modifiers (for type members)

- private: When "private" specifier is mentioned before the member field/method.
  - The members are accessible in current class (member OR this.member).
- default: When no specifier is mentioned before the member field/method.
  - The members are accessible in current class (member OR this.member).
  - The members are accessible in all classes in same package (obj.member OR ClassName.member).
- protected: When "protected" specifier is mentioned before the member field/method.
  - The members are accessible in current class (member OR this.member).
  - The members are accessible in all classes in same package including its sub-classes (super.member, obj.member OR ClassName.member).
  - The members are accessible in sub classes outside the package including its sub-classes (super.member).
- public: When "public" specifier is mentioned before the member field/method.
  - The members are accessible in current class (member OR this.member).
  - The members are accessible in all other classes (super.member, obj.member OR ClassName.member).
- Scopes

   - private (lowest)
   - default
   - protected
   - public (highest)

| | default | private | protected | public |
|---|---|---|---|---|
| same class | yes | yes | yes | yes |
| same package subclass | yes | no | yes | yes |
| same package non-subclass | yes | no | yes | yes |
| different package subclass | no | no | yes | yes |
| different package non-subclass | no | no | no | yes |

-

# Array

- Array is collection of similar data elements. Each element is accessible using indexes (0 to n-1).
- In Java, array is non-primitive/reference type i.e. its object is created using new operator (on heap).
- Array size is fixed - given while creating array object. It cannot be modified later.
- Java array object holds its length and data elements.
- The array of primitive type holds values (0 if uninitialized) and array of non-primitive type holds references (null if uninitialized).
- Array of primitive types

```
int[] arr1 = new int[5];

int[] arr2 = new int[5] { 11, 22, 33 };
// error

int[] arr3 = new int[] { 11, 22, 33, 44, 55 };

int[] arr4 = { 11, 22, 33, 44 };
```

- Array of non-primitive types

```
Human[] arr5 = new Human[5];

Human[] arr6 = new Human[] {h1, h2, h3};
// h1, h2, h3 are Human references

Human[] arr7 = new Human[] {
```

```
        new Human(...),
        new Human(...),
        new Human(...)
};
```

- In Java, checking array bounds is responsibility of JVM. When invalid index is accessed, ArrayIndexOutOfBoundsException is thrown.
- Array types are
    - 1-D array
    - 2-D/Multi-dimensional array
    - Ragged array

## 1-D array

```
int[] arr = new int[5];

int[] arr = new int[4] { 10, 20, 30, 40 };
// error

int[] arr = new int[] { 11, 22, 33, 44, 55 };

int[] arr = { 11, 22, 33, 44 };

Human[] arr = new Human[5];

Human[] arr = new Human[] {h1, h2, h3};
```

- Individual element is accesses as `arr[i]`.

###Assignments

1. Write a class TemperatureConverter with methods `double fahrenheitToCelsius(double f)` and `void celsiusToFahrenheit(double c)` in a package `weather`. Create an unpackaged Program class with main() method. Accept temparature from user and display the converted result. Keep source code in "src" directory. Create compiled packages in "bin" directory.

    - Fahrenheit To Celsius: (32°F − 32) × 5/9 = 0°C
    - Celsius To Fahrenheit: (0°C × 9/5) + 32 = 32°F

2. Create an array of 100 integers. Initialize the elements using random values between 0 to 99 using Java's Random class. Write a static method that sums all elements of array and return total. Hint:

    ```
    Random r = new Random();
    int num = r.nextInt(100);
    // arr[i] = num;
    ```