## Core Java

# Agenda

- Multi-Threading
- Nested/Inner classes

### Multi-Threading

#### Inter-thread communication

- wait()
  - Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.
  - The current thread must own this object's monitor i.e. wait() must be called within synchronized block/method.
  - The thread releases ownership of this monitor and waits until another thread notifies.
  - The thread then waits until it can re-obtain ownership of the monitor and resumes execution.
- notify()
  - Wakes up a single thread that is waiting on this object's monitor.
  - If multiple threads are waiting on this object, one of them is chosen to be awakened arbitrarily.
  - The awakened thread will not be able to proceed until the current thread relinquishes the lock on this object.
  - This method should only be called by a thread that is the owner of this object's monitor.
- notifyAll()
  - Wakes up all threads that are waiting on this object's monitor.
  - The awakened threads will not be able to proceed until the current thread relinquishes the lock on this object.
  - This method should only be called by a thread that is the owner of this object's monitor.

### Member/Nested classes

- By default all Java classes are top-level.
- In Java, classes can be written inside another class/method. They are Member classes.
- Four types of member/nested classes
  - o Static member classes -- demo11\_01
  - o Non-static member class -- demo11\_02
  - o Local class -- demo11\_03
  - o Annoymous Inner class -- demo11\_04
- When .java file is compiled, separate .class file created for outer class as well as inner class.

#### Static member classes

- Like other static members of the class (belong to the class, not the object).
- Accessed using outer class (Doesn't need the object of outer class).
- Can access static (private/public) members of the outer class directly.

- Static member class cannot access non-static members of outer class directly.
- The outer class can access all members (including private) of inner class directly (no need of getter/setter).
- The static member classes can be private, public, protected, or default.

```
class Outer {
    private int nonStaticField = 10;
    private static int staticField = 20;
    public static class Inner {
        public void display() {
            System.out.println("Outer.nonStaticField = " + nonStaticField);
// error
            System.out.println("Outer.staticField = " + staticField); // ok
- 20
        }
    }
}
public class Main {
    public static void main(String[] args) {
        Outer.Inner obj = new Outer.Inner();
        obj.display();
    }
}
```

### Non-static member classes/Inner classes

- Like other non-static members of the class (belong to the object/instance of Outer class).
- Accessed using outer class object (Object of outer class is MUST).
- Can access static & non-static (private) members of the outer class directly.
- The outer class can access all members (including private) of inner class directly (no need of getter/setter).
- The non-static member classes can be private, public, protected, or default.

```
public class Main {
   public static void main(String[] args) {
        //Outer.Inner obj = new Outer.Inner(); // compiler error
        // create object of inner class
        //Outer outObj = new Outer();
        //Outer.Inner obj = outObj.new Inner();
      Outer.Inner obj = new Outer().new Inner();
      obj.display();
   }
}
```

• If Inner class member has same name as of outer class member, it shadows (hides) the outer class member. Such Outer class members can be accessed explicitly using Outer.this.

Static member class and Non-static member class -- Application

```
// top-level class
class LinkedList {
   // static member class
   static class Node {
        private int data;
        private Node next;
        // ...
    }
    private Node head;
    // non-static member class
    class Iterator {
        private Node trav;
        // ...
    }
    // ...
    public void display() {
        Node trav = head;
        while(trav != null) {
            System.out.println(trav.data);
            trav = trav.next;
        }
    }
}
```

#### Local class

- Like local variables of a method.
- The class scope is limited to the enclosing method.
- If enclosed in static method, behaves like static member class. If enclosed in non-static method, behaves like non-static member class.
- Along with Outer class members, it can also access (effectively) final local variables of the enclosing method.
- We can create any number of objects of local classes within the enclosing method.

```
public class Main {
    private int nonStaticField = 10;
    private static int staticField = 20;
    public static void main(String[] args) {
        final int localVar1 = 1;
        int localVar2 = 2;
        int localVar3 = 3;
        localVar3++;
        // local class (in static method) -- behave like static member class
        class Inner {
            public void display() {
                System.out.println("Outer.nonStaticField = " +
nonStaticField); // error
                System.out.println("Outer.staticField = " + staticField); //
ok 20
                System.out.println("Main.localVar1 = " + localVar1); // ok 1
                System.out.println("Main.localVar2 = " + localVar2); // ok 2
                System.out.println("Main.localVar3 = " + localVar3); //
error
            }
        Inner obj = new Inner();
        obj.display();
        //new Inner().display();
    }
}
```

### Annoymous Inner class

- Creates a new class inherited from the given class/interface and its object is created.
- If in static context, behaves like static member class. If in non-static context, behaves like non-static member class.
- Along with Outer class members, it can also access (effectively) final local variables of the enclosing method.

```
// (named) local class
class EmpnoComparator implements Comparator<Employee> {
   public int compare(Employee e1, Employee e2) {
      return e1.getEmpno() - e2.getEmpno();
   }
}
Arrays.sort(arr, new EmpnoComparator()); // anonymous obj of local class
```

```
// Anonymous inner class
Comparator<Employee> cmp = new Comparator<Employee>() {
   public int compare(Employee e1, Employee e2) {
```

```
return e1.getEmpno() - e2.getEmpno();
}
};
Arrays.sort(arr, cmp);
```

# Date and Time API

- Legacy Java classes
  - o Date
  - o Calendar
- Java 8 Date Time
  - https://www.baeldung.com/java-8-date-time-intro