# Core Java

## Day 13-B Agenda

- javap tool
- Annoymous Inner classes
- Java 8 Interfaces
  - default methods
  - static methods
  - functional interfaces

## javap tool

- Part of JDK.
- To inspect the class byte-code and metadata.
- Usage:
  - javap classname
    - Display public/protected members of the class.
  - javap -p classname
    - Also display private members of the class.
  - javap -v classname
    - Display class metadata, constant pool, byte-code, etc.

## 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);
```

```
// Anonymous object of Anonymous inner class.
Arrays.sort(arr, new Comparator<Employee>() {
    public int compare(Employee e1, Employee e2) {
        return e1.getEmpno() - e2.getEmpno();
    }
});
```

## Java 8 Interfaces

- Before Java 8 --
  - Interfaces are used to design specification/standards. It contains only declarations – public abstract.

```
interface Geometry {
    /*public static final*/ double PI = 3.14;
    /*public abstract*/ int calcRectArea(int length, int breadth);
    /*public abstract*/ int calcRectPeri(int length, int breadth);
}
```

    - As interfaces doesn't contain method implementations, multiple interface inheritance is supported (no ambiguity error).
    - Interfaces are immutable. One should not modify interface once published.
- Java 8 added many new features in interfaces in order to support functional programming in Java. Many of these features also contradicts earlier Java/OOP concepts.

## Default methods

- Java 8 allows default methods in interfaces. If method is not overridden, its default implementation in interface is considered.
- This allows adding new functionalities into existing interfaces without breaking old implementations e.g. Collection, Comparator, ...

```
interface Shape {
    /*public static final*/ double PI = 3.14;
    /*public abstract*/ double calcArea();

    /*public*/ default double calcPeri() {
        return 0.0;
    }
}
```

```
class Square implements Shape {
    private double side;
    // ...
    public double calcArea() {
        return side * side;
    }
    public double calcPeri() {
        return 4 * side;
    }
}
```

```
class Rectangle implements Shape {
    private int length, breadth;
    // ...
    public double calcArea() {
        return length * breadth;
    }
}
```

```
class Circle implements Shape {
    private double radius;
    // ...
    public double calcArea() {
        return Shape.PI * radius * radius;
    }
}
```

```
// in main()
Square s = new Square(10);
System.out.println("Square Area: " + s.calcArea()); // 100
System.out.println("Square Peri: " + s.calcPeri()); // 40

Circle c = new Circle(7);
System.out.println("Circle Area: " + c.calcArea()); // ~154
```

```
Rectangle r = new Rectangle(10, 4);
System.out.println("Rectangle Area: " + r.calcArea()); // 40
System.out.println("Rectangle Peri: " + r.calcPeri()); // 0
```

## Static methods

- Before Java 8, interfaces allowed public static final fields.
- Java 8 also allows the static methods in interfaces.
- They act as helper methods and thus eliminates need of helper classes like Collections, ...

```
interface Shape {
    /*public static final*/ double PI = 3.14;
    /*public abstract*/ double calcArea();

    /*public*/ default double calcPeri() {
        return 0.0;
    }

    /*public*/ static double calcTotalArea(Shape[] arr) {
        double total = 0.0;
        for(Shape sh : arr) {
            double area = sh.calcArea();
            total = total + area;
        }
        return total;
    }
}
```

```
// in main()
Shape[] arr = new Shape[3];
arr[0] = new Square(5.0);
```

```
    arr[1] = new Circle(7.0);
    arr[2] = new Rectangle(4, 3);
    double total = Shape.calcTotalArea(arr);
    System.out.println("Total Area: " + total);
```

## Assignments

7. Create an interface Emp with abstract method `double getSal()` and a default method `default double calcIncentives()`. The default method simply returns 0.0. Create a class Manager (with fields basicSalary and dearanceAllowance) inherited from Emp. In this class override getSal() method (basicSalary + dearanceAllowance) as well as calcIncentives() method (20% of basicSalary). Create another class Labor (with fields hours and rate) inherited from Emp interface. In this class override getSal() method (hours * rate) as well as calcIncentives() method (5% of salary if hours > 300, otherwise no incentives). Create another class Clerk (with field salary) inherited from Emp interface. In this class override getSal() method (salary). Do not override, calcIncentives() in Clerk class. In Emp interface create a static method `static double calcTotalIncome(Emp arr[])` that calculate total income (salary + incentives) of all employees in the given array.