Nested Classes

Types of Nested Classes

- nested class is a class defined within another class

1. INNER CLASS

   non-static type, defined at the member level of a class

2. STATIC NESTED CLASS

   static type, defined at the member level of a class

3. LOCAL CLASS

   a class defined within a method body

4. ANONYMOUS CLASS

   local class which doesn't have a name

Inner Class

- can have access modifier
- can extend another class and/or implement interfaces
- can be marked `abstract` or `final`
- can accesss **all** members of the enclosing class

  (including private members)

```
public class A {
  private String str = "Hello!";
  protected class B {
    private static void printOut(String s) {
       System.out.print(s);
    }
    public void printHelloTwice() {
      printOut(str);
      printOut(str);
    }
  }
  public void printTwice() {
    var b = new B();
    b.printHelloTwice();
  }
  public static void main(String[] args) {
    var a = new A();
    a.printTwice();
  }
}
```

visible to inner class

inner class

creating the instance and then calling the method

```
// if method is not static you have to make an instance before calling it


public static void main(String[] args) {
  var a = new A();
  var b = a.new B();
  b.printHelloTwice();
}


// even shorter (but more uglier)
public static void main(String[] args) {
  new A().new B().printHelloTwice();
}
```

```java
// nested classes can have their own nested classes
public class A {
  private int t = 1;
  class B {
    private int t = 2;
    class C {
      private int t = 3;
      public void printT() {
        System.out.println(t);
        System.out.println(this.t);
        System.out.println(B.this.t);
        System.out.println(A.this.t);
      }
    }
  }
// main method
}
```

```java
public static void main(String[] args) {
  // initialize all classes
  A a = new A();
  B b = a.new B();
  B.C c = b.new C();
  c.printT();
}
```

```
3
3
2
1
```

Static Nested Class

- can't access instance variables or methods declared in the outer class
- you don't need an instance of the wider class to access it
- can be marked `private` or `protected`

```
public class State {
  static class Town {
    private int type = 1;
  }


  public static void main(String[] args) {
    Town town = new Town();
    System.out.println(town.type);
  }
}
```

if not static: `Town town = new State().new Town();`

```
1
```

Local Class

- nested class defined within the method
  - limited scope
- don't have access modifier
- can be declared `abstract` or `final`
- can access all members of the enclosing class
- can access `final` and effectively final **local** variables

```
// calculate and print area of the rectangle
public class PrintArea {
  private int a = 10;
  public void calculateArea() {
    final int b = 15;
    class Computer {
      public void multiply() {
        System.out.println(a*b);
      }
    }
    var computer = new Computer();
    computer.multiply();
  }
  public static void main(String[] args) {
    var printArea = new PrintArea();
    printArea.calculateArea();
  }
}
```

local class

goes out of scope when we exit the method

NOTE: class `Computer` can access both a and b

=> because <u>b is `final`</u>

(without keyword `final` b would be effectively final and therefore still accessible by `Computer` class)

Anonymous Class

- special type of **local** class which doesn't have a name
- must extend an existing class or implement an existing interface

```
public class Store {
  abstract class Sale {
    abstract int discount();
  }
  public int newPrice(int oldPrice) {
    Sale sale = new Sale () {
       int discount() { return 2; }
    };
    return oldPrice - sale.discount();
  }
}
```

`sale` is an instance of the anonymous class `{ ... }` with the implementation of `discount()` method from abstract class `Sale`

```
// it works with interfaces as well
public class Store {
  interface Sale {
    int discount();
  }
  public int newPrice(int oldPrice) {
    Sale sale = new Sale () {
      public int discount() { return 2; }
    };
    return oldPrice - sale.discount();
  }
}
```

implementation of interface method must be public !!

```
// exam trick: "empty interface"
public class Dog {
  interface Eats {}
  Eats eating = new Eats {}; OK
}

// eating is not an instance of the interface (not allowed!),
// but the instance of the anonymous class {}; implementing the interface
```