Programming Fundamentals

Inner Classes & Enumeration

Week 8| Iresh Bandara

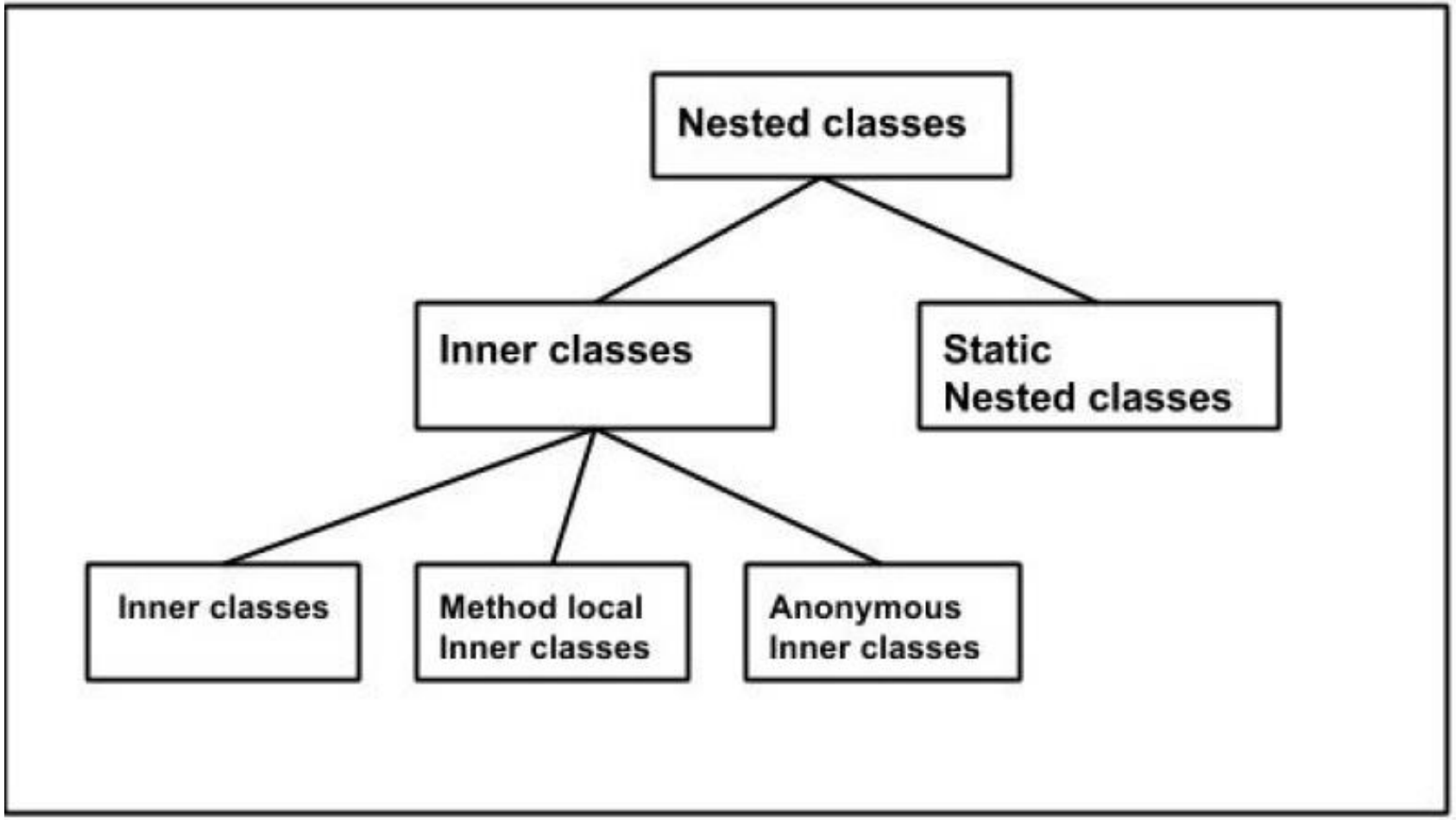
ROBERT GORDON UNIVERSITY ABERDEEN
TEF Gold
INFORMATICS INSTITUTE OF TECHNOLOGY

Learning Outcomes

- Covers part of LO2, LO4 for Module

- On completion of this lecture, students are expected to be able to:
  - Make use of the inner classes need in a java program.
  - Develop safer readable Java programs with enumeration.

Inner Classes



Inner Classes

- Nested classes are divided into two types:
  - **Non-static nested classes:** These are the non-static members of a class.
  - **Static nested classes:** These are the static members of a class.

Regular Inner Class

```
class Outer{
     class Inner{ }
}
```

- **Compilation**
  - **javac Outer.java**
- **Ouput ByteCodes**
  - **Outer.class**
  - **Outer$Inner.class**

Private members

- Inner class can have access to Outer classes private members

```
class Outer {
private int x = 7;
     class Inner {
     public void seeOuter() {
     System.out.println(x);
     }
}
}
```

Running the inner-class

- You can't access the Inner class in a usual way
  - Using java Outer$Inner
- because regular Inner-class can not contain
  - main-method.
  - It cannot contain any static content

Inner Class

- Creating an inner class is quite simple. You just need to write a class within a class. Unlike a class,
- an inner class can be private and once you declare an inner class private, it cannot be accessed
- from an object outside the class.
- Given below is the program to create an inner class and access it. In the given example, we make
- the inner class private and access the class through a method

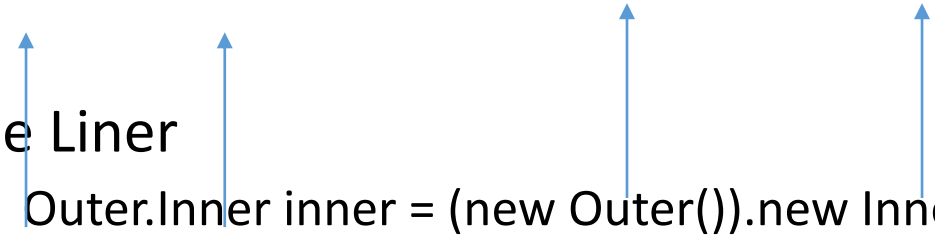
Instantiating an Inner class

- To instantiate: You must have an instance of the outer class
- Instantiating an Inner Class from within Outer classes

```
class Outer{
private int x = 7;
public void makeInner(){
Inner x = new Inner();
x.seeOuter();
}
class Inner{
public void seeOuter(){
System.out.println(x);
}
}
}
```

Creating an Inner Class from Outside of the Outer class

- You have to have instance of the Outer-class
  - Outer outer = new Outer();
- After that, you create the Inner object
  - Outer.Inner inner = outer.new Inner()



  - One Liner
    - Outer.Inner inner = (new Outer()).new Inner();

Referencing Inner / Outer class from Inner class

```
class Outer{
class Inner{
public void seeOuter(){
System.out.println(this);
System.out.println(Outer.this);
}
}
}
```

Method-local Inner Classes

- Class inside a method
- Can be instantiated only within the method (below the class)
- Can use Outer classes private members
- Cannot use methods variables
  - Unless the variable is final

Example

```
class Outer{
private int x = 7;
public void method(){
final String y = "hi!";
String z = "hi!";
class Inner{
public void seeOuter(){
System.out.println(x); // works!!
System.out.println(y); // works!
//System.out.println(z); // doesn't work!
}
}
Inner object = new Inner();!
object.seeOuter();
}
}
```

Anonymous Inner Classes

```
class Popcorn {
public void pop() {
System.out.println("popcorn");
}
}
class Food {!
        Popcorn p = new Popcorn() {
        public void pop() {
                System.out.println("subclass Popcorn!");
        }
};//Notice the semicolon!

}
```

Static Nested Classes

- Static member of the enclosing class.

```
 class BigOuter{
static class Nested { }
 }
```

- Does not have access to the instance variables

```
BigOuter.Nested n = new
BigOuter.Nested()
```

Enums

- An enum type is a special data type that enables for a variable to be a set of predefined constants. The variable must be equal to one of the values that have been predefined for it. Common examples include compass directions (values of NORTH, SOUTH, EAST, and WEST) and the days of the week.

- Because they are constants, the names of an enum type's fields are in uppercase letters.

- In the Java programming language, you define an enum type by using the enum keyword. For example, you would specify a days-of-the-week enum type as:

```
public enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURDAY
}
```

Anti-pattern: int constants

```
public class Card {
    public static final int CLUBS = 0;
    public static final int DIAMONDS = 1;
    public static final int HEARTS = 2;
    public static final int SPADES = 3;

    ...
    private int suit;
    ...
    public void setSuit(int suit) {
        this.suit = suit;
    }
}
```

- What's wrong with using `int` constants to represent card suits?
  - variation (also bad): using `Strings` for the same purpose.

Enumerated types

- **enum**: A type of objects with a fixed set of constant values.

```
public enum Name {
    VALUE, VALUE, ..., VALUE
}
```

  - Usually placed into its own .java file.
  - C has `enum`s that are really `int`s; Java's are objects.

```
public enum Suit {
    CLUBS, DIAMONDS, HEARTS, SPADES
}
```

- **Effective Java Tip :** Use `enum`s instead of `int` constants.

  "The advantages of `enum` types over `int` constants are compelling. Enums are far more readable, safer, and more powerful."

What is an enum?

- The preceding `enum` is roughly equal to the following short class:

```
public final class Suit extends Enum<Suit> {
    public static final Suit CLUBS    = new Suit();
    public static final Suit DIAMONDS = new Suit();
    public static final Suit HEARTS   = new Suit();
    public static final Suit SPADES   = new Suit();

    private Suit() {}   // no more can be made
}
```

What can you do with an enum?

- use it as the type of a variable, field, parameter, or return

```
public class Card {
    private Suit suit;
    ...
}
```

- compare them with `==` (why don't we need to use `equals`?)

```
if (suit == Suit.CLUBS) { ...
```

- compare them with `compareTo` (by order of declaration)

```
public int compareTo(Card other) {
    if (suit != other.suit) {
        return suit.compareTo(other.suit);
    } ...
}
```

The switch statement

```
switch (boolean test) {
    case value:
        code;
        break;
    case value:
        code;
        break;
    …
    default:  // if it isn't one of the above values
        code;
        break;
}
```

- an alternative to the `if/else` statement
  - must be used on integral types (e.g. `int`, `char`, `long`, **`enum`**)
  - instead of a `break`, a case can end with a `return`, or if neither is present, it will "fall through" into the code for the next case

Enum methods

method	description
`int compareTo(E)`	all `enum` types are Comparable by order of declaration
`boolean equals(o)`	not needed; can just use `==`
`String name()`	equivalent to `toString`
`int ordinal()`	returns an enum's 0-based number by order of declaration (first is 0, then 1, then 2, ...)

method	description
`static E valueOf(s)`	converts a string into an `enum` value
`static E[] values()`	an array of all values of your enumeration

EnumSet

- class `EnumSet` from `java.util` represents a set of `enum` values and has useful methods for manipulating `enum`s:

`static EnumSet<E> allOf(Type)`	a set of all values of the type
`static EnumSet<E> complementOf(set)`	a set of all enum values other than the ones in the given set
`static EnumSet<E> noneOf(Type)`	an empty set of the given type
`static EnumSet<E> of(...)`	a set holding the given values
`static EnumSet<E> range(from, to)`	set of all enum values declared between from and to

```
Set<Coin> coins = EnumSet.range(Coin.NICKEL, Coin.QUARTER);
for (coin c : coins) {
    System.out.println(c);              // see also: EnumMap
}
```

- **Effective Java Tip :** Use `EnumSet` instead of bit fields.
- **Effective Java Tip :** Use `EnumMap` instead of ordinal indexing.

More complex enums

- An enumerated type can have fields, methods, and constructors:

```
public enum Coin {
    PENNY(1), NICKEL(5), DIME(10), QUARTER(25);

    private int cents;

    private Coin(int cents) {
        this.cents = cents;
    }

    public int getCents()  { return cents; }

    public int perDollar() { return 100 / cents; }

    public String toString() {        // "NICKEL (5c)"
        return super.toString() + " (" + cents + "c)";
    }
}
```

Summery

- Nested classes are divided into two types:Non-static nested classes and Static nested classes.
- Inner class can have access to Outer classes private members.
- An enum type is a special data type that enables for a variable to be a set of predefined constants.
- Enums are far more readable, safer, and more powerful.

Thank you

ROBERT GORDON UNIVERSITY ABERDEEN
TEF Gold
INFORMATICS INSTITUTE OF TECHNOLOGY