

Java Programming Language SE - 6

Module 7 : Advanced Class Features

Team Emertxe

ORACLE®

Certified Professional

Java SE 6 Programmer



Objectives

- Create static variables, methods, and initializers
- Create final classes, methods, and variables
- Create and use enumerated types
- Use the static import statement
- Create abstract classes and methods
- Create and use an interface





Relevance



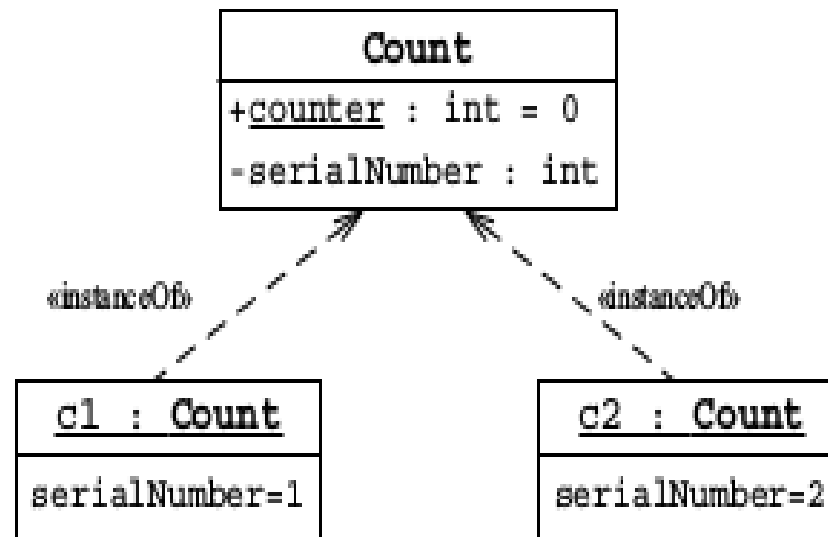
- How can you create a constant?
- How can you declare data that is shared by all instances of a given class?
- How can you keep a class or method from being subclassed or overridden?

The static Keyword

- The static keyword is used as a modifier on variables, methods, and nested classes.
- The static keyword declares the attribute or method is associated with the class as a whole rather than any particular instance of that class.
- Thus static members are often called class members, such as class attributes or class methods.

Class Attributes/ Static Variables

- Class attributes are shared among all instances of a class.
- it can be accessed without an instance.



Class Methods/ Static Methods



```
public static int getTotalCount() {  
    return counter;  
}
```

- You can invoke static methods without any instance of the class to which it belongs.
- Static methods cannot access instance variables.

Static Initializers



- A class can contain code in a static block that does not exist within a method body.
- Static block code executes once only, when the class is loaded.
- Usually, a static block is used to initialize static (class) attributes.

Static Initializers: Example

```
static {  
    counter = Integer.getInteger("myApp.Count4.counter").intValue();  
}
```


The final Keyword

- You cannot subclass a final class.
- You cannot override a final method.
- A final variable is a constant.
- You can set a final variable once only, but that assignment can occur independently of the declaration; this is called a *blank final variable*.
- A blank final instance attribute must be set in every constructor.
- A blank final method variable must be set in the method body before being used.

Final Variables

Constants are *static final* variables.

```
public class Bank {  
    private static final double  
    ... // more declarations  
}
```



Blank Final Variables

```
private final long customerId;  
public Customer() {  
    customerId = createID();  
}
```

Enumerated Type

```
package cards.domain;  
  
public enum Suit {  
  
    SPADES,  
  
    HEARTS,  
  
    CLUBS,  
  
    DIAMONDS  
  
}
```

Enumerated Type: Example

```
package cards.domain;

public class PlayingCard {
    private Suit suit;
    private int rank;
    public PlayingCard(Suit suit, int rank) {
        this.suit = suit;
        this.rank = rank;
    }
    public Suit getSuit() {
        return suit;
    }
}
```

Enumerated Type

```
public String getSuitName() {  
    String name = "";  
    switch ( suit ) {  
        case SPADES:  
            name = "Spades";  
            break;  
        case HEARTS:  
            name = "Hearts";  
            break;  
        case CLUBS:  
            name = "Clubs";  
            break;  
        case DIAMONDS:  
            name = "Diamonds";  
            break;  
        default:  
    }return name;}  
}
```

Enumerated Type

- Enumerated types are type-safe:

```
package cards.tests;
import cards.domain.PlayingCard;
import cards.domain.Suit;
public class TestPlayingCard {
    public static void main(String[] args) {
        PlayingCard card1 = new PlayingCard(Suit.SPADES, 2);
        System.out.println("card1 is the " + card1.getRank() + " of " + card1.getSuitName());
        // PlayingCard card2 = new PlayingCard(47, 2);
        // This will not compile.
    }
}
```

Advanced Enumerated Types



- Enumerated types can have attributes and methods:

```
package cards.domain;

public enum Suit {
    SPADES
    ("Spades"),
    HEARTS
    ("Hearts"),
    CLUBS
    ("Clubs"),
    DIAMONDS ("Diamonds");
    private final String name;
    private Suit(String name) {
        this.name = name;
    }
    public String getName() {
        return name;}}}
```


Advanced Enumerated Types



```
package cards.tests;
import cards.domain.PlayingCard;
import cards.domain.Suit;
public class TestPlayingCard {
    public static void main(String[] args) {
        PlayingCard card1 = new PlayingCard(Suit.SPADES, 2);
        System.out.println("card1 is the " + card1.getRank()
            + " of " + card1.getSuit().getName());
        // NewPlayingCard card2 = new NewPlayingCard(47, 2);
        // This will not compile.
    }
}
```

Static Imports

- A static import imports the static members from a class:

```
import static <pkg_list>.<class_name>.<member_name>;
```

OR

```
import static <pkg_list>.<class_name>.*;
```

- A static import imports members individually or collectively:

```
import static cards.domain.Suit.SPADES;
```

OR

```
import static cards.domain.Suit.*;
```

Abstract Classes

```
public class FuelNeedsReport {  
    private Company company;  
    public FuelNeedsReport(Company company) {  
        this.company = company;  
    }  
    public void generateText(PrintStream output) {  
        Vehicle1 v;  
        double fuel;  
        double total_fuel = 0.0;  
        for ( int i = 0; i < company.getFleetSize(); i++ ) {  
            v = company.getVehicle(i);
```

Abstract Classes

```
// Calculate the fuel needed for this trip
fuel = v.calcTripDistance() / v.calcFuelEfficiency();
output.println("Vehicle " + v.getName() + " needs "
+ fuel + " liters of fuel.");
total_fuel += fuel;
}
output.println("Total fuel needs is " + total_fuel + " liters.");
}}
```

Abstract Classes

- An abstract class models a class of objects in which the full implementation is not known but is supplied by the concrete subclasses.

Interfaces

- A public interface is a contract between client code and the class that implements that interface.
- A Java interface is a formal declaration of such a contract in which all methods contain no implementation.
- Many unrelated classes can implement the same interface.
- A class can implement many unrelated interfaces.
- Syntax of a Java class is as follows:

```
<modifier> class <name> [extends <superclass>]  
[implements <interface> [,<interface>]* ] {  
  <member_declaration>*  
}
```

Uses of Interfaces

Interface uses include the following:

- Declaring methods that one or more classes are expected to implement
- Determining an object's programming interface without revealing the actual body of the class
- Capturing similarities between unrelated classes without forcing a class relationship
- Simulating multiple inheritance by declaring a class that implements several interfaces

Stay connected

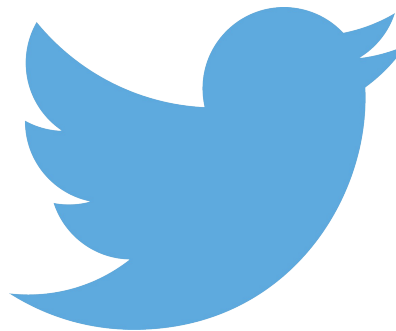


About us: Emertxe is India's one of the top IT finishing schools & self learning kits provider. Our primary focus is on Embedded with diversification focus on Java, Oracle and Android areas

Emertxe Information Technologies,
No-1, 9th Cross, 5th Main,
Jayamahal Extension,
Bangalore, Karnataka 560046
T: +91 80 6562 9666
E: training@emertxe.com



<https://www.facebook.com/Emertxe>



<https://twitter.com/EmertxeTweet>



slideshare
Present Yourself

<https://www.slideshare.net/EmertxeSlides>



Thank You