1495 UNIVERSITY OF ABERDEEN

# JC2002 Java Programming

## Lecture 8: Enum types, static, and final

# Enum types and keywords static and final

- What are enum types?
  - Enum declaration
- Keyword static
  - Static class members
  - Static import
- Keyword final
  - Principle of least privilege
  - Final instance variables
- Much of the material is based on slides from ***Java: How to Program***, chapter 8, which is available via MyAberdeen

# What are enum types?

- Like classes, all `enum` types are reference types
- The basic `enum` type defines a set of constants represented as unique identifiers
- For every `enum`, the compiler generates the `static` method **`values()`** that returns an array of the `enum`'s constants
- The `enum` constants can be used anywhere constants can be used, such as in the `case` labels of `switch` statements and to control enhanced `for` statements

# Enum declaration

- An enum type is declared with an *enum declaration*, which is a *comma-separated* list of enum *constants*
- The declaration may optionally include other components of traditional classes, such as constructors, fields and methods
  - An enum constructor can specify any number of parameters and it can be overloaded
- Each enum declaration declares an enum class with the following restrictions:
  - Enum constants are implicitly `final` and `static`
  - Any attempt to create an object of an enum type with operator new results in a compilation error

# Enum declaration example

Book.java

```java
public enum Book {
  // declare constants of enum type
  JHTP("Java How to Program", "2018"),
  CHTP("C How to Program", "2016"),
  IW3HTP("Internet & World Wide Web How to Program", "2012"),
  CPPHTP("C++ How to Program", "2017"),
  VBHTP("Visual Basic How to Program", "2014"),
  CSHARPHTP("Visual C# How to Program", "2017");

  // instance fields
  private final String title;
  private final String copyrightYear;

 // enum constructor
  Book(String title, String copyrightYear) {
    this.title = title;
    this.copyrightYear = copyrightYear;
  }
```

```java
  // accessor for field title
  public String getTitle() {
    return title;
  }
  // accessor for field copyrightYear
  public String getCopyrightYear() {
    return copyrightYear;
  }
}
```

# Enum methods

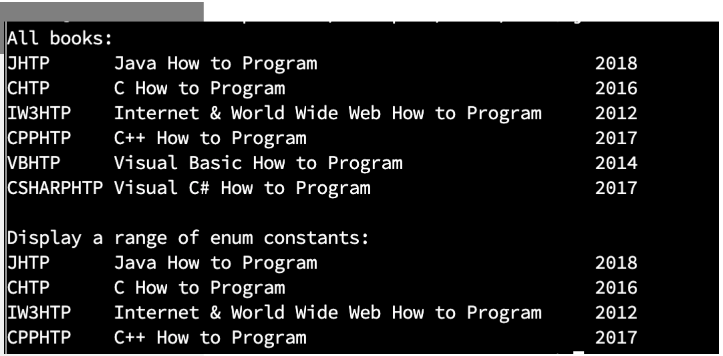
- The enhanced `for` statement can be used with an `EnumSet` just as it can with an array
- Method **`range()`** of class **`EnumSet`** (declared in package `java.util`) can be used to access a range of an `enum`'s constants
  - Method `range` takes two parameters: the first and the last enum constant in the range
  - Returns an `EnumSet` that contains all the constants between these two constants, both inclusive
- Class `EnumSet` provides several other `static` methods

# Enum usage example

EnumTest.java

```java
import java.util.EnumSet;

public class EnumTest {
  public static void main(String[] args) {
    System.out.println("All books:");
    // print all books in enum Book
    for (Book book : Book.values()) {
      System.out.printf("%-10s%-45s%s%n", book,
        book.getTitle(), book.getCopyrightYear());
    }
    System.out.printf("%nDisplay a range of enum constants:%n");
    // print first four books
    for (Book book : EnumSet.range(Book.JHTP, Book.CPPHTP)) {
      System.out.printf("%-10s%-45s%s%n", book,
        book.getTitle(), book.getCopyrightYear());
    }
  }
}
```

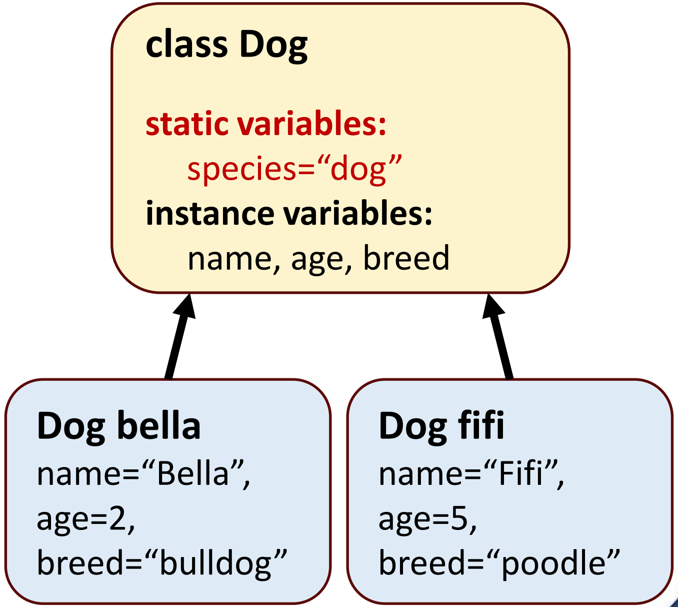


```
All books:
JHTP      Java How to Program                          2018
CHTP      C How to Program                             2016
IW3HTP    Internet & World Wide Web How to Program     2012
CPPHTP    C++ How to Program                           2017
VBHTP     Visual Basic How to Program                  2014
CSHARPHTP Visual C# How to Program                     2017

Display a range of enum constants:
JHTP      Java How to Program                          2018
CHTP      C How to Program                             2016
IW3HTP    Internet & World Wide Web How to Program     2012
CPPHTP    C++ How to Program                           2017
```

# Static class members

- A `static` field (called a *class variable*) is used in the case of only one copy of a particular variable should be *shared* by all objects of a class
- A `static` variable have *class scope*, which represents *class-wide* information: all objects of the class share the *same* piece of data, and it can also be used in all of the class's methods
- The declaration of a `static` variable begins with the keyword `static`



**class Dog**

**static variables:**
species="dog"
**instance variables:**
name, age, breed

**Dog bella**
name="Bella",
age=2,
breed="bulldog"

**Dog fifi**
name="Fifi",
age=5,
breed="poodle"

# Features of static class members

- Static class members are available as soon as the class is loaded into memory at execution time
  - Class members declared as `private static` can be accessed by client code only through methods of the class
  - A class's `public static` members can be accessed through a reference to any object of the class, or by qualifying the member name with the class name and a dot (`.`), as in `Math.random()`
- When no objects of the class exist:
  - To access a `public static` member, prefix the class name and a dot (`.`) to the `static` member, as in `Math.PI`
  - To access a `private static` member, provide a `public static` method and call it by qualifying its name with the class name and a dot

# Features of static methods

- Since a `static` method can be called even when no objects of the class have been instantiated, a `static` method *cannot* access a class's instance variables and instance methods
  - The `this` reference *cannot* be used in a `static` method: the `this` reference must refer to a specific object of the class, but when a `static` method is called, there might not be any objects of its class in memory
- If a `static` variable is not initialized, the compiler assigns it a default value (e.g., the default value for type `int` is `0`)

# Static class member example (1)

Employee.java

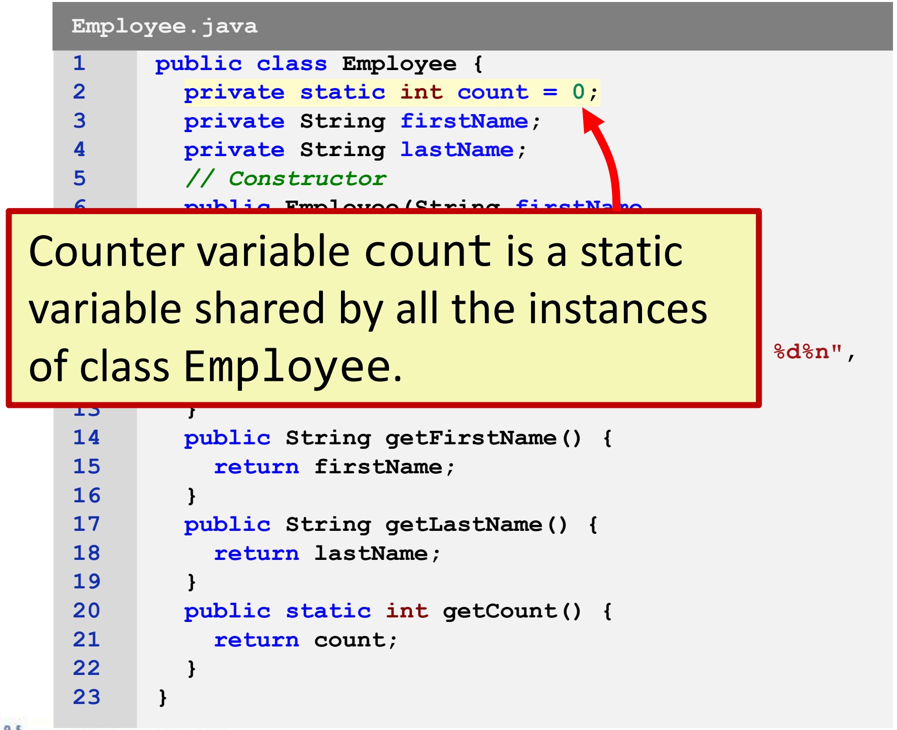
```java
public class Employee {
  private static int count = 0;
  private String firstName;
  private String lastName;
  // Constructor
  public Employee(String firstName,
      String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
    ++count; // increment static count
    System.out.printf("Name %s %s; count = %d%n",
        firstName, lastName, count);
  }
  public String getFirstName() {
    return firstName;
  }
  public String getLastName() {
    return lastName;
  }
  public static int getCount() {
    return count;
  }
}
```

EmployeeTest.java

```java
public class EmployeeTest {
  public static void main(String[] args) {
  System.out.printf("Employees before: %d\n",
      Employee.getCount());
  // create two Employees; count should be 2
  Employee e1 = new Employee("Susan", "Baker");
  Employee e2 = new Employee("Bob", "Blue");

  // show that count is now 2
  System.out.printf("\nEmployees after:\n");
  System.out.printf("via e1.getCount(): %d\n",
      e1.getCount());
  System.out.printf("via e2.getCount(): %d\n",
      e2.getCount());
  System.out.printf("via Employee.getCount(): %d\n",
      Employee.getCount());
  // get names of Employees
  System.out.printf("\nEmployee 1: %s %s%n",
      e1.getFirstName(), e1.getLastName());
  System.out.printf("\nEmployee 2: %s %s%n",
      e2.getFirstName(), e2.getLastName());
  }
}
```

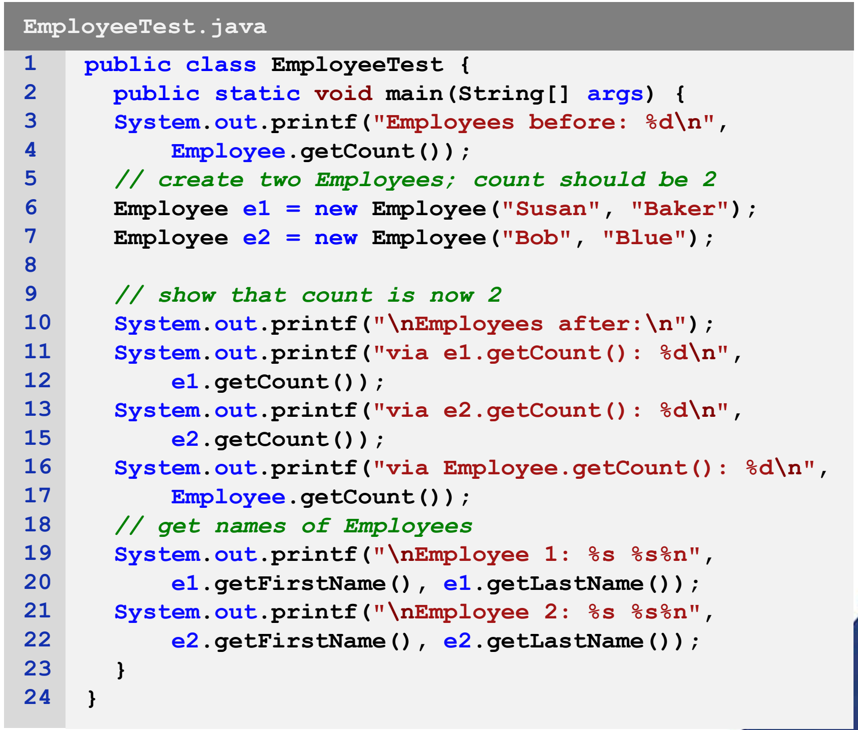
# Static class member example (2)



Employee.java

```java
public class Employee {
  private static int count = 0;
  private String firstName;
  private String lastName;
  // Constructor
  public Employee(String firstName
                                                    %d%n",
  }
  public String getFirstName() {
    return firstName;
  }
  public String getLastName() {
    return lastName;
  }
  public static int getCount() {
    return count;
  }
}
```

Counter variable count is a static variable shared by all the instances of class Employee.



EmployeeTest.java

```java
public class EmployeeTest {
  public static void main(String[] args) {
  System.out.printf("Employees before: %d\n",
      Employee.getCount());
  // create two Employees; count should be 2
  Employee e1 = new Employee("Susan", "Baker");
  Employee e2 = new Employee("Bob", "Blue");

  // show that count is now 2
  System.out.printf("\nEmployees after:\n");
  System.out.printf("via e1.getCount(): %d\n",
      e1.getCount());
  System.out.printf("via e2.getCount(): %d\n",
      e2.getCount());
  System.out.printf("via Employee.getCount(): %d\n",
      Employee.getCount());
  // get names of Employees
  System.out.printf("\nEmployee 1: %s %s%n",
      e1.getFirstName(), e1.getLastName());
  System.out.printf("\nEmployee 2: %s %s%n",
      e2.getFirstName(), e2.getLastName());
  }
}
```

# Static class member example (3)

Employee.java

```java
public class Employee {
 private static int count = 0;
 private String firstName;
 private String lastName;
 // Constructor
 public Employee(String firstName,
   String lastName) {
  this.firstName = firstName;
  this.lastName = lastName;
  ++count; // increment static count
  System.out.printf("Name %s %s; count = %d%n",
    firstName, lastName, count);
 }
```

```
Employees before: 0
```

```java
 public String getLastName() {
  return lastName;
 }
 public static int getCount() {
  return count;
 }
}
```

EmployeeTest.java

```java
public class EmployeeTest {
 public static void main(String[] args) {
 System.out.printf("Employees before: %d\n",
   Employee.getCount());
 // create two Employees; count should be 2
 Employee e1 = new Employee("Susan", "Baker");
 Employee e2 = new Employee("Bob", "Blue");

 // show that count is now 2
 System.out.printf("\nEmployees after:\n");
 System.out.printf("via e1.getCount(): %d\n",
   e1.getCount());
 System.out.printf("via e2.getCount(): %d\n",
   e2.getCount());
 System.out.printf("via Employee.getCount(): %d\n",
   Employee.getCount());
 // get names of Employees
 System.out.printf("\nEmployee 1: %s %s%n",
   e1.getFirstName(), e1.getLastName());
 System.out.printf("\nEmployee 2: %s %s%n",
   e2.getFirstName(), e2.getLastName());
 }
}
```

# Static class member example (4)

Employee.java

```java
public class Employee {
  private static int count = 0;
  private String firstName;
  private String lastName;
  // Constructor
  public Employee(String firstName,
      String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
    ++count; // increment static count
    System.out.printf("Name %s %s; count = %d%n",
        firstName, lastName, count);
  }
```

```
Employees before: 0
Name: Susan Baker; count = 1
Name: Bob Blue; count = 2
```

```java
  }
  public static int getCount() {
    return count;
  }
}
```

EmployeeTest.java

```java
public class EmployeeTest {
  public static void main(String[] args) {
  System.out.printf("Employees before: %d\n",
      Employee.getCount());
  // create two Employees; count should be 2
  Employee e1 = new Employee("Susan", "Baker");
  Employee e2 = new Employee("Bob", "Blue");

  // show that count is now 2
  System.out.printf("\nEmployees after:\n");
  System.out.printf("via e1.getCount(): %d\n",
      e1.getCount());
  System.out.printf("via e2.getCount(): %d\n",
      e2.getCount());
  System.out.printf("via Employee.getCount(): %d\n",
      Employee.getCount());
  // get names of Employees
  System.out.printf("\nEmployee 1: %s %s%n",
      e1.getFirstName(), e1.getLastName());
  System.out.printf("\nEmployee 2: %s %s%n",
      e2.getFirstName(), e2.getLastName());
  }
}
```

# Static class member example (5)

Employee.java

```java
public class Employee {
  private static int count = 0;
  private String firstName;
  private String lastName;
  // Constructor
  public Employee(String firstName,
      String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
    ++count; // increment static count
    System.out.printf("Name %s %s; count = %d%n",
        firstName, lastName, count);
  }
```

```
Employees before: 0
Name: Susan Baker; count = 1
Name: Bob Blue; count = 2

Employees after:
via e1.getCount(): 2
via e2.getCount(): 2
via Employee.getCount(): 2
```

EmployeeTest.java

```java
public class EmployeeTest {
  public static void main(String[] args) {
  System.out.printf("Employees before: %d\n",
      Employee.getCount());
  // create two Employees; count should be 2
  Employee e1 = new Employee("Susan", "Baker");
  Employee e2 = new Employee("Bob", "Blue");

  // show that count is now 2
  System.out.printf("\nEmployees after:\n");
  System.out.printf("via e1.getCount(): %d\n",
      e1.getCount());
  System.out.printf("via e2.getCount(): %d\n",
      e2.getCount());
  System.out.printf("via Employee.getCount(): %d\n",
      Employee.getCount());
  // get names of Employees
  System.out.printf("\nEmployee 1: %s %s%n",
      e1.getFirstName(), e1.getLastName());
  System.out.printf("\nEmployee 2: %s %s%n",
      e2.getFirstName(), e2.getLastName());
  }
}
```

# Static class member example (6)

Employee.java

```java
public class Employee {
  private static int count = 0;
  private String firstName;
  private String lastName;
  // Constructor
  public Employee(String firstName,
      String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
    ++count; // increment static count
```

```
Employees before: 0
Name: Susan Baker; count = 1
Name: Bob Blue; count = 2

Employees after:
via e1.getCount(): 2
via e2.getCount(): 2
via Employee.getCount(): 2

Employee 1: Susan Baker
Employee 2: Bob Blue
```

EmployeeTest.java

```java
public class EmployeeTest {
  public static void main(String[] args) {
  System.out.printf("Employees before: %d\n",
      Employee.getCount());
  // create two Employees; count should be 2
  Employee e1 = new Employee("Susan", "Baker");
  Employee e2 = new Employee("Bob", "Blue");

  // show that count is now 2
  System.out.printf("\nEmployees after:\n");
  System.out.printf("via e1.getCount(): %d\n",
      e1.getCount());
  System.out.printf("via e2.getCount(): %d\n",
      e2.getCount());
  System.out.printf("via Employee.getCount(): %d\n",
      Employee.getCount());
  // get names of Employees
  System.out.printf("\nEmployee 1: %s %s%n",
      e1.getFirstName(), e1.getLastName());
  System.out.printf("\nEmployee 2: %s %s%n",
      e2.getFirstName(), e2.getLastName());
  }
}
```

# Static import

- A *static import* declaration enables you to import the `static` members of a class or interface so you can access them via their *unqualified names* in your class. i.e., the class name and a dot (`.`) are *not* required when using an imported `static` member
- Two forms of static import:
  - One that imports a particular `static` member (which is known as *single static import*)
  - One that imports all `static` members of a class (which is known as *static import on demand*)

# Static import syntax

- The following syntax imports a particular `static` member:

  **`import static`** ***`packageName.ClassName.staticMemberName;`***

- The following syntax imports *all* `static` members of a class:

  **`import static`** ***`packageName.ClassName.*;`***

  - where *`packageName`* is the package of the class, *`ClassName`* is the name of the class and *`staticMemberName`* is the name of the `static` field or method
  - Wildcard * indicates that *all* `static` members of the specified class should be imported

- Note that static import declarations import only `static` class members: Regular `import` statements should be used to specify the classes used in a program

# Static import example

```java
// Static import of Math class methods.
import static java.lang.Math.*;

public class StaticImportTest {
    public static void main(String[] args) {
        System.out.printf("sqrt(900.0) = %.1f\n", sqrt(900.0));
        System.out.printf("ceil(-9.8) = %.1f\n", ceil(-9.8));
        System.out.printf("E = %f\n", E);
        System.out.printf("PI = %f\n", PI);
    }
}
```

```
sqrt(900.0) = 30.0
ceil(-9.8) = -9.0
E = 2.718282
PI = 3.141593
```

# Final instance variables

- Keyword `final` specifies that a variable is not modifiable (i.e., it is a constant) and any attempt to modify it gives an error
    - A `final` variable cannot be modified by assignment after it has been initialized
    - A `final` variable can be initialised when is declared, e.g., to declare a `final` (constant) instance variable `INCREMENT` of type `int`, use:

      ```
      private final int INCREMENT;
      ```

    - Different objects of the class can have different value for the `final` variable, if it is initialised with a different value in different constructors of the class

# Why to use final variables?

- The *principle of least privilege* is fundamental to good software engineering
  - Code should be granted only the amount of privilege and access that it needs to accomplish its designated task, but no more
  - This principle makes your programs more robust by preventing code from accidentally (or maliciously) modifying variable values and calling methods that should not be accessible

# Questions, comments?

1495
UNIVERSITY OF
ABERDEEN