

The logo for Oracle Academy is centered on a light gray background. It features the word "ORACLE" in a bold, orange, sans-serif font. Below it, the word "Academy" is written in a smaller, dark gray, sans-serif font. The entire logo is framed by a thin black border, with dark gray horizontal bars at the top and bottom.

# ORACLE

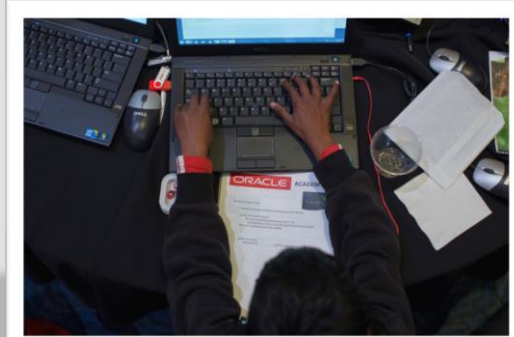
## Academy

# Java Fundamentals

7-3

## The Static Modifier and Nested Classes

**ORACLE**  
Academy



Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

# Objectives

- This lesson covers the following objectives:
  - Create static variables
  - Use static variables
  - Create static methods
  - Use static methods
  - Create static classes
  - Use static classes



# Static Modifier

- Using instance variables, each instance of a class created with the keyword `new` creates a copy of all instance variables in that class
- For example, in the `Employee` class below, a unique copy of `lastname` and `firstname` is created for each new `Employee` object that is created in a `Driver Class`

```
public class Employee{
    private String lastname;
    private String firstname;
    ...more code
} //end class Employee
//create two Employees in a main method:
Employee e1 = new Employee("Smith", "Mary");
Employee e2 = new Employee("Jones", "Sally");
```

It is common practice to have the fields as `private` to protect them from direct access. We would provide accessor and mutator methods for access to these fields if required.

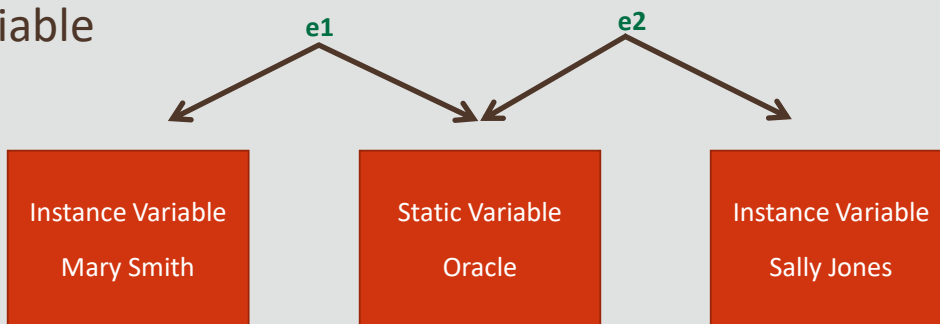
# Static Keyword

- Static is a keyword in Java that modifies the association of an item to a class
- Contents of a class that are identified as static are shared across all instances of the class
- This means all instances of the class share one copy of the static items, and each have their own unique copies of instance items, or non-static items

When we add the static modifier to a field/variable then this is normally referred to as being a class variable. Where the value is only stored once and is accessible by all the class instances.

## Static Example

- Consider initializing a static String with the value "Oracle" called myCompany that represents the employer's company
- Each instance of Employee would still have their unique instance variables, but would share the static variable



So changing the text in the name field of e1 would only effect that one instance. If we updated e1 for myCompany then as every other instance of Employee references the same value then they would also see this change.

# Static Variables

- Static variables:
  - Are also known as class variables
  - Are declared with the static keyword
  - Have only one copy in memory, as opposed to instance variables, which hold one copy per instance
  - Are shared by object instances
  - Hold the same value for all class instances

Static values are useful for tasks such as Keeping count of how many objects are created, when you only need one copy of useful constants or common values that will be used across multiple objects, etc.)

# Static Variables

- Public access for static variables:
  - If public, they can be modified directly by other classes
  - Consider making the variable a constant by using the keyword `final` to prevent modifications
  - Example:

```
public static final int MODEL_NUM = 883;
```

Even though there is a Java keyword `const`, it is not used. `final` is used for constants in Java.



# Programming Practices and Static Variables

- Good programming practice initializes static variables with values, rather than relying on the default null and 0 values
- The values initially assigned can be changed as long as the class is active in JVM memory
- Garbage collection removes it from memory and the initial values assigned will return the next time you use it

## Declaring a Static Variable

- To declare a static variable, include the keyword `static` as shown below
  - Can be public, protected, default, or private
  - Should have assigned values, but automatically are assigned null values for class instances: an empty string or 0 for primitive numbers
  - Should act as constants with the `final` keyword when they use a public, protected, or default access

```
public class Nesting {  
    // Declare public static variable.  
    public static final int MODEL_NUM = 883;  
    ...  
} //end class Nesting
```

The final keyword does not have to be used with static variables. This simply forces that the variable cannot be changed once a value has been set.

## Changes to a Static Variable

- Static variables that are not final can be read or assigned new values by using the optional keyword `this` in instance methods
  - Changes by instance methods are changed for all instances
  - A change to a static variable may indicate that the class should be limited to only one object
  - This is known as the Singleton pattern

```
private static String myCompany = "Oracle";
public void setMyCompany(String s) {
    this.mycompany = s;
} //end method myCompany
...
```

The Singleton pattern forces the idea that there can only be one instance of a class. On the first call to this class a new instance is created. All future calls will not create new instances, but return the one that was created.

# Static Variable Example

- Create a class called Turtle that contains a variable named food
  - This variable is static since all of our turtles eat the same food
  - The Turtle class will have one more variable named age
  - Since each turtle is a different age, it is best to make this variable a private instance variable rather than a static one

```
public class Turtle {  
    public static String food = "Turtle Food";  
    private int age;  
  
    public Turtle(int age) {  
        this.age = age;  
    } //end constructor  
} //end class Turtle
```



## Accessing Static Variables

- Instance variables require an instance of the class to exist before access is possible

```
public class Turtle {  
    public static String food = "Turtle Feed";  
    private int age;  
    ...  
} //end class Turtle
```

- You can access static variables without creating an instance of the class
- In a main method, this statement would print out the variable food without any instance reference

```
System.out.println("I feed " + Turtle.food  
    + " to all of my turtles!");
```

In fact, good programming practices say that you should always use the class name to access static variables and not the name of an instance.

## Notation to Access Static Variables

- Generally, static variables are accessed by the notation:

```
ClassName.variableName;
```

# Static Modifier and Methods

- Static or class methods exist once and are shared by all class instances
  - May be used by other class methods or instance methods based on their access modifier
  - Cannot access non-static, or instance, variables
  - Static methods can only access static variables
  - Cannot access non-static, or instance, methods
  - Static methods can only access other static methods
  - Can be redefined in subclasses
  - Can be public, protected, default, or private

## Static Modifier and Methods

- There are differences between calling an instance method versus a class (static) method
- For example
  - You must first create an instance and then use a dot notation to call an instance method; whereas, the class name, a dot notation, and static method name calls a static method



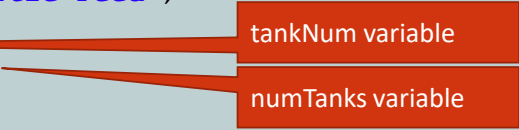
## Static Modifier and Methods

- The static method provides a wrapper to construct an instance of a class
- When the class has a private access constructor, a static method is one of two approaches to creating an instance of the class

## Turtle Class Example

- The Turtle class has a static variable that identifies the number of tanks we have available (numTanks) and an instance variable (tankNum) that tells us which tank the Turtle is in

```
public class Turtle {  
    public static String food = "Turtle Feed";  
    public int age;  
    public int tankNum;  
    public static int numTanks = 3;  
    public Turtle(int age) {  
        this.age = age;  
        tankNum = (int) ((Math.random() * numTanks) + 1);  
    } //end constructor  
    public void swim() { //implementation  
    public int getAge() { return age; }  
    public int getTankOfResidence() { return tankNum; }  
    public static String fishTank() { return "I have " + numTanks  
                                         + " fish tanks.";  
    } //end method fishTank  
} //end class Nesting
```



# Static Methods in Turtle Class Example



- Review the methods in the Turtle class

```
public class Turtle {  
    public static String food = "Turtle Feed";  
    public int age;  
    public int tankNum;  
    public static int numTanks = 3;  
    public Turtle(int age){  
        this.age = age;  
        tankNum = (int)((Math.random()*numTanks)+1);  
    } //end constructor  
    public void swim(){//implementation}  
    public int getAge(){return age;}  
    public int getTankOfResidence(){return tankNum;}  
    public static String fishTank() {return "I have " + numTanks + " fish tanks.";}  
    } //end method fishTank  
} //end class Turtle
```

swim() is a instance method. Although each turtle can swim, the turtles may swim differently depending on their age.

fishTank() is a static method and it accesses a static variable (numTanks).

getAge() and getTankOfResidence() are instance, non-static, methods because they access non-static variables. Static methods cannot access non-static items.

# Creating Class Instances Using Static Methods

- Another use of static methods is for creating class instances when the class constructor access is private, and the method is part of the same class
- This is possible because the static method is publicly accessible with private access to the class

```
...  
private Nesting() {...implementation...}  
...  
public static Nesting getInstance() {  
    Nesting nesting = new Nesting();  
    return nesting; }  
...  
// Instantiate a private class with a method.  
Nesting n1 = Nesting.getInstance();  
...
```

# Static Modifier and Classes

- Static or nested classes:
  - Can exist as nested classes
  - Cannot exist as independent classes



A nested class is a class that is created inside another class.



**ORACLE**  
Academy

JF 7-3  
The Static Modifier and Nested Classes

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

21

There are a few other ways of nesting classes in Java that are not covered in this course.

# Static Nested Classes

- Static nested classes
  - Are implemented inside other classes, and the other classes are known as container classes
  - Can extend the behavior of the container class
  - Can be overloaded like ordinary constructors

## Static Nested Classes

- The static nested class also provides the means for instantiating a containing class when its constructor is configured with private access
- This is the second way to instantiate a class that has a restricted or private access qualifier for its class constructors

# Static Nested Classes Example

```
public class Space {  
    //Space class variables  
  
    public static class Planet{  
        //planet class variables and constructors  
        public Planet() {  
            ...implementation...  
        }//end constructor  
  
        public Planet(String, int size){  
            ...implementation...  
        }//end constructor  
    }//end class Planet  
  
    //more space class implementation  
}//end class Space
```



# Terminology

- Key terms used in this lesson included:
  - Class method
  - Class variable
  - Inner class
  - Nested class
  - Static modifier
  - Static method
  - Static nested class
  - Static variable

# Summary

- In this lesson, you should have learned how to:
  - Create static variables
  - Use static variables
  - Create static methods
  - Use static methods
  - Create static classes
  - Use static classes



The logo for Oracle Academy. The word "ORACLE" is in a bold, orange, sans-serif font. Below it, the word "Academy" is in a smaller, dark gray, sans-serif font. The entire logo is centered on a light gray background, which is framed by dark gray horizontal bars at the top and bottom.

# ORACLE

## Academy