ORACLE® ACADEMY

Java Fundamentals

7-1
Classes, Objects, and Methods

ORACLE ACADEMY

Objectives

This lesson covers the following objectives:

- Recognize the correct general form of a class
- Create an object of a class
- Create methods that compile with no errors
- Return a value from a method
- Use parameters in a method
- Create a driver class and add instances of Object classes

Objectives

This lesson covers the following objectives:

- Add a constructor to a class
- Apply the new operator
- Describe garbage collection and finalizers
- Apply the this reference
- Add a constructor to initialize a value

Creating a Class Template

- Programmers can create their own classes.
- Classes are essentially a template or blueprint for all instances of the class.
- The class code also communicates to the compiler how to define, create, and interact with objects of the class.
- The code on the following slide starts to create the Class Vehicle which will represent the basic outline for Vehicle objects.

Creating a Class Template Example

```
public class Vehicle {
        // the Vehicle class has two fields
          private String make;
          private int milesPerGallon;

        //constructor
          public Vehicle(){
          }
       //mutator/setter method
          public void setMake(String m){
             make = m;
          }
      //mutator/setter method
          public void setMilesPerGallon(int mpg){
             milesPerGallon = mpg;
         }
      //accessor/getter method
          public String getMake(){
             return make;
          }
       //accessor/getter method
          public int getMilesPerGallon(){
             return milesPerGallon;
         }
}
```

Creating an Instance of a Class

- Once you have created a class, you can create instances of the class (objects) in a Driver Class or inside other Object Classes.
- Instances:
  – Inherit all attributes and methods defined in the class template.
  – Interact independently of one another.
  – Are reference objects.
  – Are created using the new operator.

Instantiate an Instance

- To instantiate an instance of a Vehicle called myCar, write:

```
public class VehicleTester{
    public static void main(String[] args){
        Vehicle myCar = new Vehicle();
    }
}
```

In Java, instantiation is the creation of objects from a class.

Constructors

- Constructors are methods that allow the user to create instances of (instantiate) a class.
- Good programming practice dictates that classes should have a default constructor.
- Constructors which contain parameters typically initialize the private variables of the class to values passed in by the user.
- Constructors do not have a return type (void or other).

Default Constructor

- Good programming practice dictates that classes should have a default constructor.
- A default constructor:
  - Takes no parameters.
  - Typically initializes all private variables to base values.

```
public Vehicle() {
    make = "";
    milesPerGallon = 0;
}
```

Constructor with Parameters

- A constructor with parameters is used when you want to initialize the private variables to values other than the default values.

```
public Vehicle(String m, int mpg){
    make=m;
    milesPerGallon=mpg;
}
```

Parameters

Parameters are variables that are listed as part of a method (or constructor) declaration. In the example above, String m and int mpg are parameters. Values are given to the parameters when a call to the method or constructor is made.

Instantiate Vehicle Instance

- To instantiate a Vehicle instance using the constructor with parameters, use arguments:

Arguments

```
Vehicle myCar = new Vehicle("Toyota", 30);
```

- To instantiate a Vehicle instance using the default constructor, write:

```
Vehicle myCar = new Vehicle();
```

Defining Methods

- A method is a block of code which is referred to by name and can be called at any point in a program simply by utilizing the method's name. There are four main parts to defining your own method:
  - Access Modifier (public, private, protected, default)
  - Return type
  - Method name
  - Parameter(s)

```
public returnType methodName(Parameter p, …)
{
 /*code that will execute with each call to the
   method goes here*/
}
```

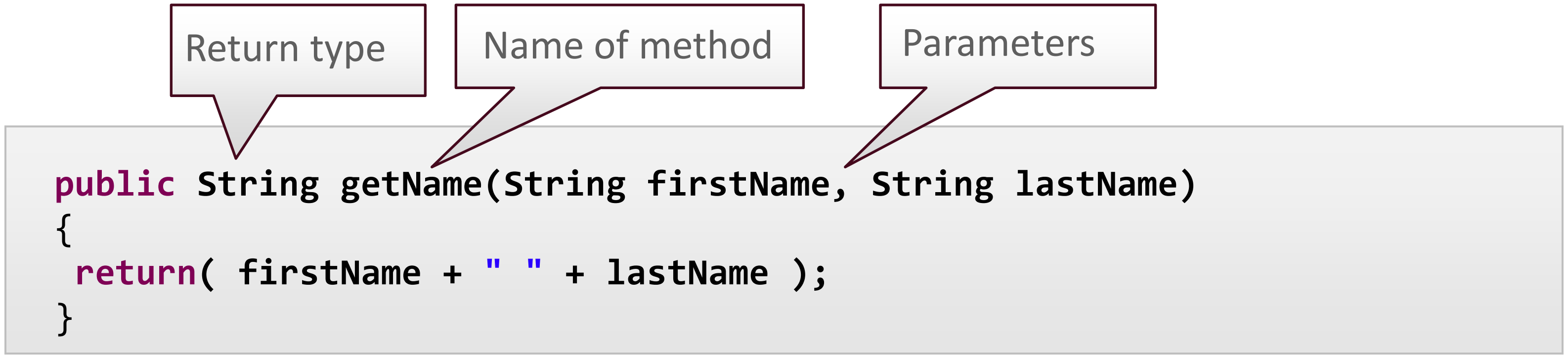
Components of a Method

- Method components include:
  - Return type:
    - This identifies what type of object, if any, will be returned when the method is invoked (called).
    - If nothing will be returned, the return type is declared as void.
  - Method name:
    - Used to make a call to the method.

Components of a Method

- Parameter(s):
  - The programmer may choose to include parameters depending on the purpose and function of the method.
  - Parameters can be of any primitive or type of object, but the parameter type used when calling the method must match the parameter type specified in the method definition.

Method Components Example



```
public String getName(String firstName, String lastName)
{
 return( firstName + " " + lastName );
}
```

Class Methods

- Every class will have a set of methods associated with it which allow functionality for the class.
- Accessor method
  - "getter"
  - Returns the value of a specific private variable.
- Mutator method
  - "setter"
  - Changes or sets the value of a specific private variable.
- Functional method
  - Returns or performs some sort of functionality for the class.

Accessor Methods

- Accessor methods access and return the value of a specific private variable of the class.
- Non-void return type corresponds to the data type of the variable you are accessing.
- Include a return statement.
- Usually have no parameters.

```
public String getMake(){
    return make;
}

public int getMilesPerGallon(){
    return milesPerGallon;
}
```

Mutator Methods

- Mutator methods set or modify the value of a specified private variable of the class.
- Void return type.
- Parameter with a type that corresponds to the type of the variable being set.

```
public void setMake(String m){
    make = m;
}

public void setMilesPerGallon(int mpg){
    milesPerGallon = mpg;
}
```

Functional Methods

- Functional methods perform a functionality for the class.
- Void or non-void return type.
- Parameters are optional and used depending on what is needed for the method's function.

Functional Methods

- Below is a functional method for the class Vehicle that compares two vehicles and returns an int value for the comparison.

```
//Compares the miles per gallon of each vehicle passed in, returns 0 if
they are the same, 1 if the first vehicle is larger than the second and -1
if the second vehicle is larger than the first*/

public int compareTo(Vehicle v1, Vehicle v2){
     if(v1.getMilesPerGallon()= = v2.getMilesPerGallon())
          return 0;
     if(v1.getMilesPerGallon()> v2.getMilesPerGallon())
          return 1;
     return -1;
}
```

Using Constructors and Methods in a Driver class main method Example

- For the following:
  - What functionality does each line have?
  - What will the final print statement print to the screen?

```
public class VehicleTester{
  public static void main(String[] args){

    Vehicle v;
     v=new Vehicle();
     v.setMake("Ford");
     v.setMilesPerGallon(35);

    System.out.print("My "+v.getMake() +
         " gets " + v.getMilesPerGallon() + " mpg.");
  }
}
```

this Reference

- Within an instance method or a constructor, **this** is a reference to the current object.
- The reference to the object whose method or constructor is being called.
- Refer to any member of the current object by using **this**.
- Most commonly used when a field is shadowed by a method or constructor parameter of the same name.

this Reference Example

- When a method argument "shadows" a field of the object, the **this** reference is used to differentiate the local scope from the class scope.

```
public class Point {
    private int x;
    Private int y;

    //constructor
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

Card Class Example

- Consider a standard deck of playing cards.
- To represent each card as an instance of a Card class, what attributes would the class need to have?
  - Suit
  - Name
  - Points

```
public class Card {
        private String suit;
        private String name;
        private int points;
}
```

Reference Object Representation

- When creating a new instance of an object, a reference is made to the object in memory.
- The reference points to the object.
- All attribute variables are created and initialized based on the constructor used.

```
Card c = new Card();
```

suit = null
name= null
points = 0

Understanding Garbage Collection Example

- Considering the code below, what will happen in memory after the line c2 = c; ?
- When executed, c2 = c; takes the reference c2 and makes it reference the same object as c.
- This effectively renders the original object c2 useless, and garbage collection takes care of it by removing it from memory.

```
Card c=new Card("Diamonds","Four", 4);
Card c2=new Card("Spades","Ace", 1);
c2 = c;
```

Finalizers

- A finalizer is code called by the garbage collector when it determines no more references to the object exist.
- All objects inherit a finalize() method from java.lang.Object.
- This method takes no parameters and is written to perform no action when called.

Finalizers

- Overriding the finalize() method in classes allows you to modify what happens before garbage collection, such as:
  - Notifying the user about the garbage collection that is about to occur.
  - Cleaning up non-Java resources, such as closing a file.

Finalize Method Example

- This is an example of the finalize() method overridden in a class. It closes all associated files and notifies the user that the finalization occurs.

```
protected void finalize(){
   try{
     close();  //close all files
   }
   finally{
     System.out.println("Finalization has occured");
   }
}
```

Terminology

Key terms used in this lesson included:

- Accessor method
- Class
- Constructor
- Finalizers
- Garbage collection
- Initialization
- Instantiate
- Method

Terminology

Key terms used in this lesson included:

- Mutator method
- new
- Null
- Object
- Reference
- this Reference

Summary

In this lesson, you should have learned how to:

- Recognize the correct general form of a class
- Create an object of a class
- Create methods that compile with no errors
- Return a value from a method
- Use parameters in a method
- Create a driver class and add instances of Object classes

Summary

In this lesson, you should have learned how to:

- Add a constructor to a class
- Apply the new operator
- Describe garbage collection and finalizers
- Apply the this reference
- Add a constructor to initialize a value

ORACLE® ACADEMY