6

Describing Objects and Classes

ORACLE

Interactive Quizzes

Before you start today's lessons, test your knowledge by answering some quiz questions that relate to yesterday's lessons. Open the Quiz files by clicking the quizzes.html shortcut from the desktop of your VM. In the welcome page, JavaSEProgrammingI.html, click the links for Lessons 2, 3 ,4, and 5.

Objectives

After completing this lesson, you should be able to:

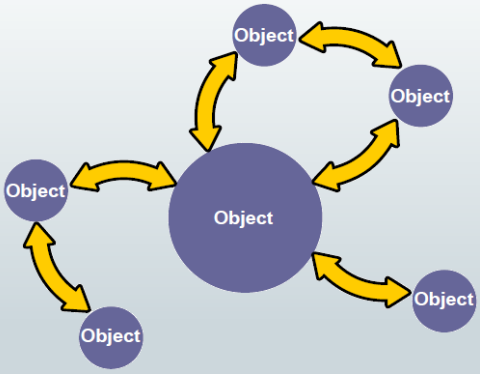
- List the characteristics of an object
- Define an object as an instance of a class
- Instantiate an object and access its fields and methods
- Describe how objects are stored in memory
- Instantiate an array of objects
- Describe how an array of objects is stored in memory
- Declare and instantiate an object as a field

Topics

- Describing objects and classes
- Defining fields and methods
- Declaring, instantiating, and using objects
- Working with object references
- Doing more with arrays
- Introducing the soccer league use case

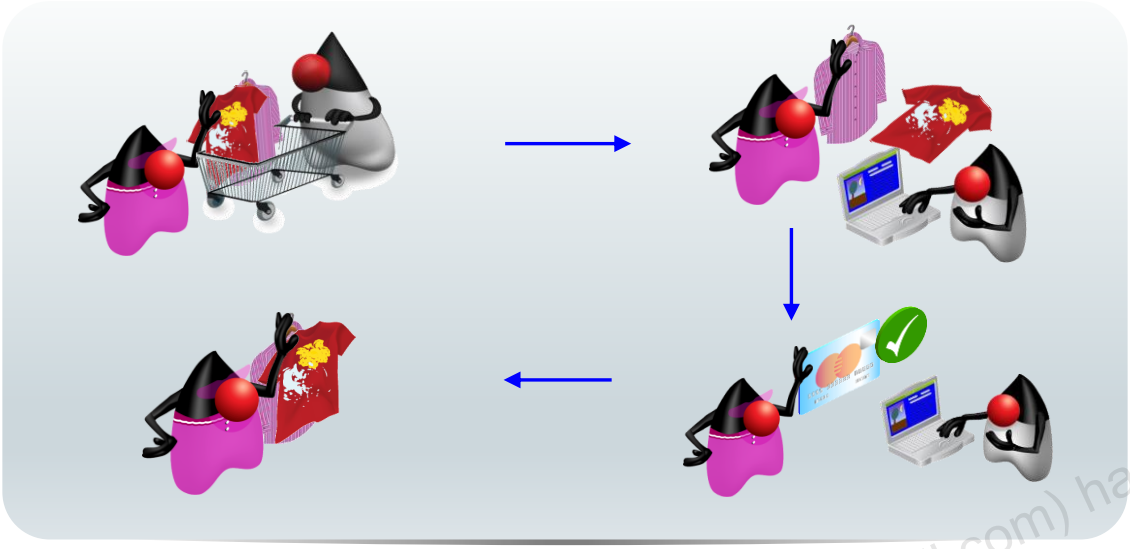
Object-Oriented Programming

- Interaction of objects
- No prescribed sequence



You have seen this diagram before in the “What Is a Java Program?” lesson. The diagram illustrates how object-oriented programming stresses the interaction of objects. The current lesson teaches you how to identify the objects that are required for the application that you would like to build. You first identify what the objects are, you determine the object’s characteristics or properties, and then you determine the object’s behaviors or operations. You then translate that analysis into Java code to create your application. It is time to learn more about objects.

Duke's Choice Order Process



In the first five lessons, the exercises mention a shopping cart class that contains items. Take another look at the shopping cart scenario.

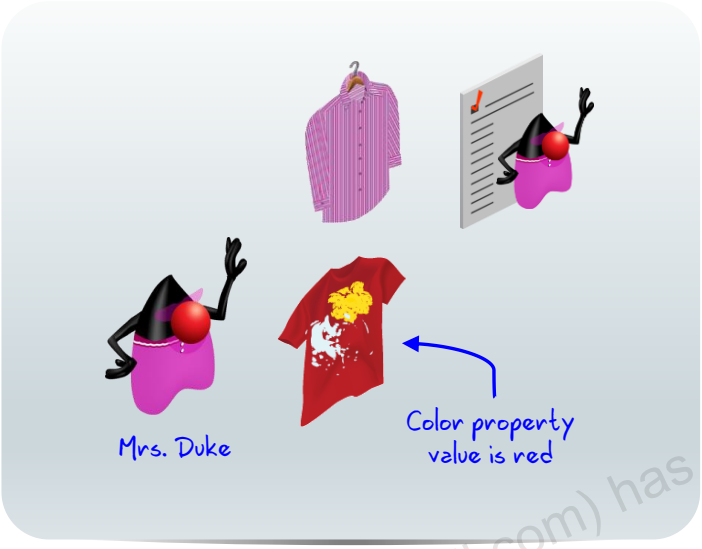
Imagine an online store called Duke's Choice. His number one shopper is his mother, Mrs. Duke. As Mrs. Duke shops, she places items in a shopping cart. Mrs. Duke likes shirts, so she places shirts in her cart. After she fills the cart, she checks out. The checkout process applies the purchase to a credit card, which is verified, and then Mrs. Duke receives an order number so that she can track her order or return it.

As a software developer, when you are presented with a scenario such as Duke's Choice for an application that you need to develop, you can analyze the scenario by breaking it into steps and defining the objects of the scenario.

Characteristics of Objects

Objects are physical or conceptual.

- Objects have **properties**:
  - Size
  - Shape
  - Name
  - Color
- Objects have **behaviors**:
  - Shop
  - Put item in cart
  - Pay

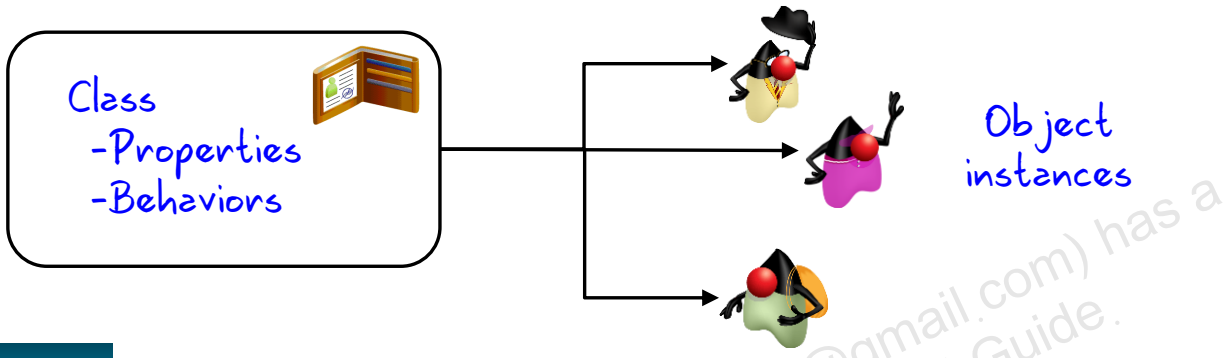


To validate objects in a problem domain, such as the Duke's Choice order process, you identify the properties of all objects:

- Objects can be physical or conceptual. A customer's credit card account is an example of a conceptual object, because it is not something you can physically touch. A shirt is an example of a physical object.
- Objects have properties (attributes) such as size, name, and shape that represent the state of the object. For example, a person has a name (Mrs. Duke), and an object might have a color property. The value of all of an object's properties is often referred to as the object's current state. An object might have a color property with the value of red and a size property with a value of large.
- Objects also have behaviors (things they can do) such as, in our example, shop, put an item in the cart, and purchase.

Classes and Instances

- A class:
  - Is a blueprint or recipe for an object
  - Describes an object's properties and behaviors
  - Is used to create object instances



You just learned about some of the objects, characteristics, and behaviors in the Duke's Choice scenario. Here is an example of one of Duke's Choice objects, the `Customer`, and its function in the store. `Customer` is the class, and a class is a blueprint or recipe for an object. The class describes an object's properties and behaviors.

Classes are used to create object instances, such as the three `Customer` object instances, as illustrated by the three images.

Quiz

Which of the following statements is true?

a. An object is a blueprint for a class.
b. An object and a class are exactly the same.
c. An object is an instance of a class.
d. A class is an instance of an object.

Answer: c

- a is false because a class is a blueprint for an object.
- b is false because an object is an instantiation of a class, and a class serves as a blueprint for the object.
- c is correct.
- d is false because an object is an instance of a class.

Topics

- Describing objects and classes
- Defining fields and methods
- Declaring, instantiating, and using objects
- Working with object references
- Doing more with arrays
- Introducing the soccer league use case

You have just learned about objects, classes, and their characteristics (properties and behaviors). Now it is time to look at fields and methods.

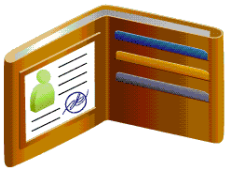
The Customer Properties and Behaviors

Properties:

- Name
- Address
- Age
- Order number
- Customer number

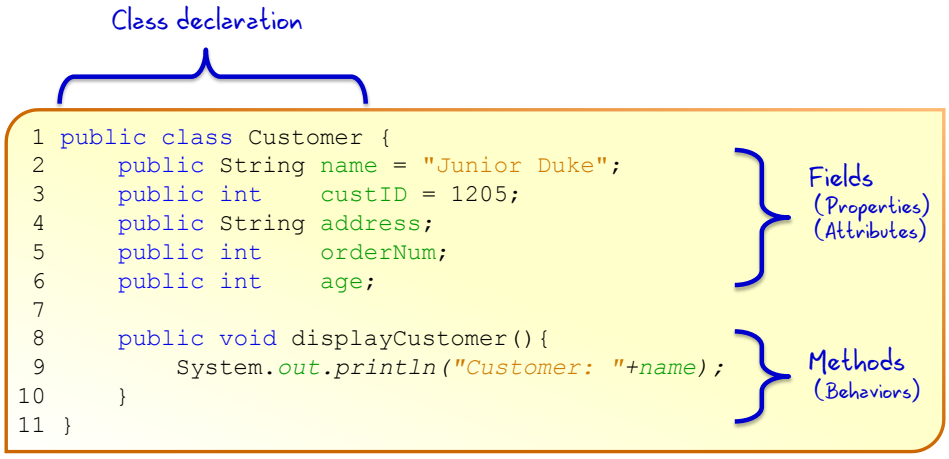
Behaviors:

- Shop
- Set Address
- Add item to cart
- Ask for a discount
- Display customer details



Think of some properties and behaviors that are in the `Customer` class of Duke's Choice. Think about how you would write this information as a Java class.

The Components of a Class

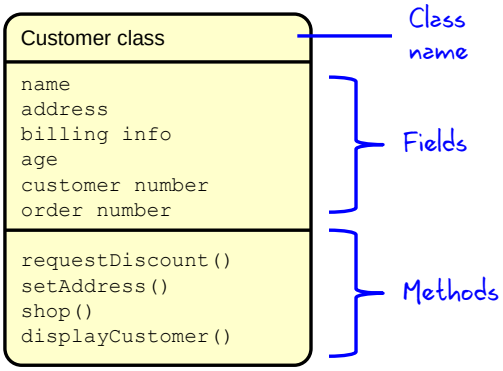


In the previous slide, you have identified some properties and behaviors that might be in the `Customer` class. This code example demonstrates how the properties and methods are created in Java. The basic components of a Java class are:

- The class declaration. Notice that the entire class is surrounded by braces.
- Fields of the class. These represent the properties or attributes of the class.
- Methods of the class. These represent the behaviors or operations. Here you see just one method, `displayCustomer`.

  Note: In the code example above, the word “public” is a modifier, and you learn about modifiers later in the course.

Modeling Properties and Behaviors



As you design an application, it is often helpful to create a simple model that describes the components of a class. In the table above, the class name is listed at the top. The properties or fields are listed in the second row, and the behaviors, or methods, are listed in the third row. If you compare this modeling in terms of language, you can think of the class as a noun, the properties or fields as adjectives, and the behaviors or methods as verbs.

Exercise 6-1: Creating the `Item` Class

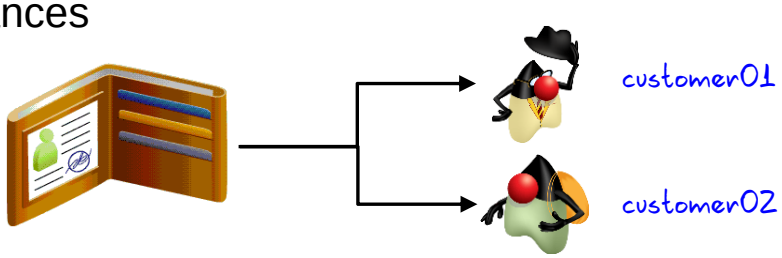
1. Open the project **`Exercise_06-1`** in NetBeans
2. Create the Item class as a plain **Java class**.
3. Declare public fields for `ID` (int), `descr` (String), `price` (double), and `quantity` (int).
   - You will not be able to test the `Item` class until Exercise 6-2.

In this exercise, you create the Item class and declare public fields.

Topics

- Describing objects and classes
- Defining fields and methods
- Declaring, instantiating, and using objects
- Working with object references
- Doing more with arrays
- Introducing the soccer league use case

Customer Instances




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

    Customer customer01 = new Customer();
    Customer customer02 = new Customer();

    customer01.age = 40;
    customer02.name = "Duke";

    customer01.displayCustomer();
    customer02.displayCustomer();
    }
}
```

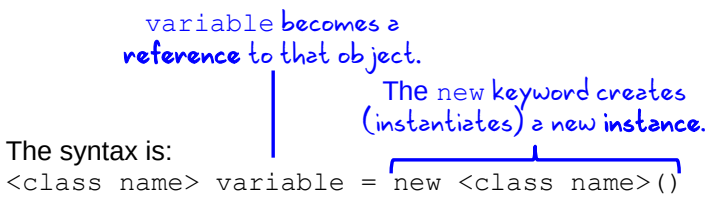
Create new instances (instantiate).

Fields are accessed.

Methods are called.

In the code example, two new instances of the `Customer` object called `customer01` and `customer02` are created. (Another term for created is “instantiated.”) After the objects are instantiated, the reference variables `customer01` and `customer02` can be used to access fields and methods of the objects. The next two slides explain variations on instantiation, and the dot operator. There is more information on methods in the lesson titled “Creating and Using Methods.”

Object Instances and Instantiation Syntax




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

    Customer customer01 = new Customer();   //Declare and instantiate

    Customer customer02;                    //Declare the reference
    customer02 = new Customer();            //Then instantiate

    new Customer();                 //Instantiation without a reference
                                    //We can't use this object later
                                    //without knowing how to reference it.
    }
}
```


By using the `new` keyword, a new instance of the class is now available to be accessed through the variable, which stores a reference to that object. It can be referred to as a reference variable or an object reference.

Notice that, following the `new` keyword, you see the class name followed by parentheses. This looks similar to calling a method, doesn't it? You are calling a method—the `constructor` method of the Customer class. Every class has a `constructor` method that has the same name as the class. Constructors are covered in more detail in the lesson titled "Creating and Using Methods."

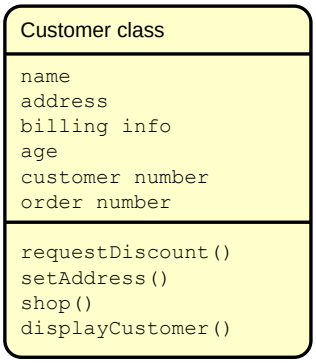
To summarize, there are three steps to getting an object reference:

1. Declare the reference.
2. Instantiate the object using the new keyword and the class `constructor` method.
3. Assign the object to the reference.

Note that the way that the assignment operator (an = symbol) works requires that the reference and the newly created object must be in the same statement. (Statements are ended with the semicolon symbol and are not the same as lines. The end of a line means nothing to the Java compiler; it only helps make the code more readable.)

The Dot (.) Operator

Follow the reference variable with a dot operator (.) to access the fields and methods of an object.



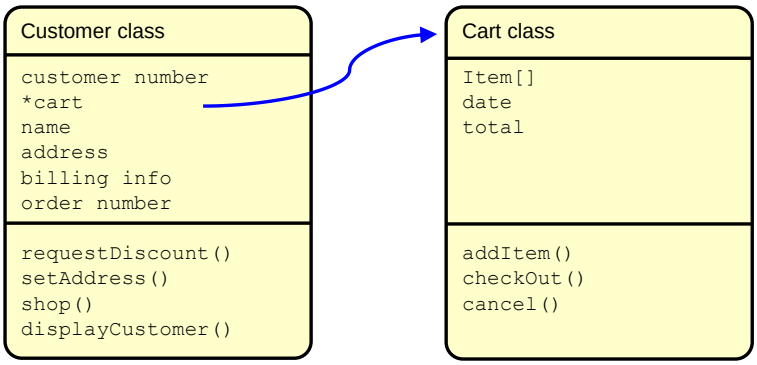

```
public static void main(String[] args){

    Customer customer01 = new Customer();

    //Accessing fields
    System.out.println(customer01.name);
    customer01.age = 40;

    //Calling methods
    customer01.requestDiscount();
    customer01.displayCustomer();
    }
}
```

Objects with Another Object as a Property




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

   Customer customer01 = new Customer();
   customer01.cart.cancel();              //How to access methods of an
                                          //object within another object
}
```


So far you have seen objects with properties such as `boolean`, `int`, `double`, and `String`. What if you wanted an object's property to be another object with its own set of properties and behaviors, such as a costumer with a cart property? That way, an instance of a `Customer` would have access to the properties and behaviors found in a `Cart`. This would enable the customer to add items to the cart and then `checkOut` (purchase) the cart. Can this be done? The answer is yes.

You can access fields and methods of objects within another object by applying the dot operator multiple times.

Note: A best practice is to use attribute and operation names that clearly describe the attribute or operation. The asterisk (*) denotes an attribute that is a reference to another object.

Quiz

Which of the following lines of code instantiates a `Boat` object and assigns it to a `sailBoat` object reference?

a. `Boat sailBoat = new Boat();`

b. `Boat sailBoat;`

c. `Boat = new Boat()`

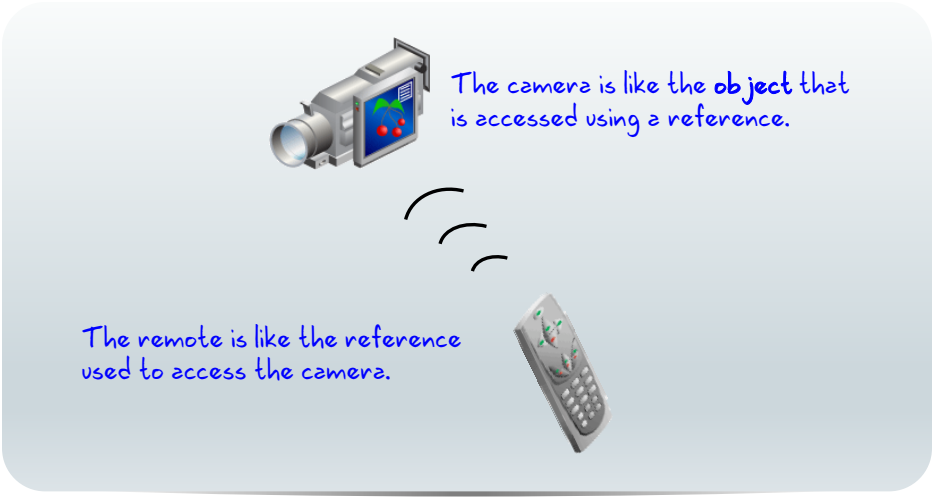
d. `Boat sailBoat = Boat();`

Answer: a

Topics

- Describing objects and classes
- Defining fields and methods
- Declaring, instantiating, and using objects
- Working with object references
- Doing more with arrays
- Introducing the soccer league use case

Accessing Objects by Using a Reference



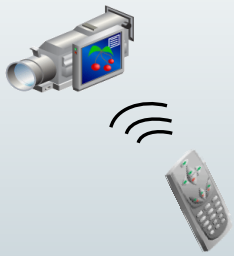
What you have learned up to this point is:

- Objects are accessed using references.
- Objects are instantiated objects of their class type.
- Objects consist of properties and operations, which in Java are fields and methods.

To work with an object, you need to access it using a reference. A good analogy is using a remote control to operate an electronic device. The buttons on the remote control can be used to modify the behavior of the device (in this case, a camera). For example, you can make the camera stop, play, or record by interacting with the remote.

Working with Object References

1. Pick up the remote to gain access to the camera.



2. Press the remote's controls to have camera do something.

1. Create a Camera object and get a reference to it.

```
11 Camera remote1;
12
13 remote1 =  new Camera();
14
15 remove1.play();
```

2. Call a method to have the Camera object do something.

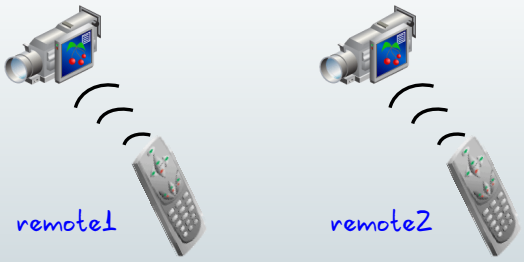
Consider the analogy of using a remote control to operate an electronic device. To operate an electronic device with a remote, you need to:

1. Pick up the remote (and possibly turn it on)
2. Press a button on the remote to do something on the camera

Similarly, to do something with a Java object, you need to:

1. Get its "remote" (called a reference)
2. Press its "buttons" (called methods)

Working with Object References

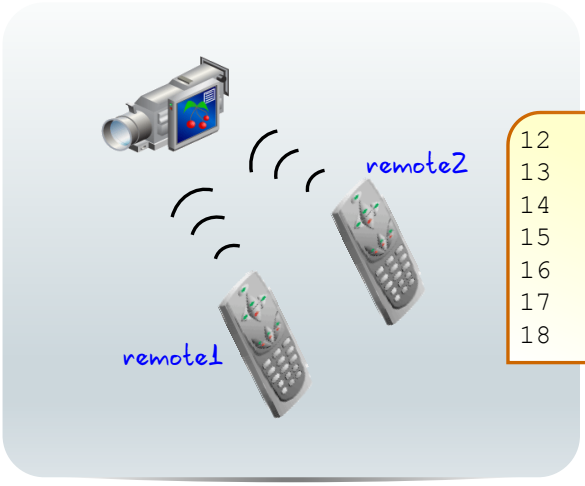



```
12  Camera remote1 = new Camera();
13
14  Camera remote2 = new Camera();
15
16  remote1.play();
17
18  remote2.play();
```

There are two Camera objects.

There are two camera objects in this example. Each camera has its own unique remote. remote2 will not work on remote1's camera, and remote1 will not work on remote2's camera. This reflects how in Java, two different objects can be instantiated with their own unique references. These references can be used to call methods on their respective objects.

Working with Object References

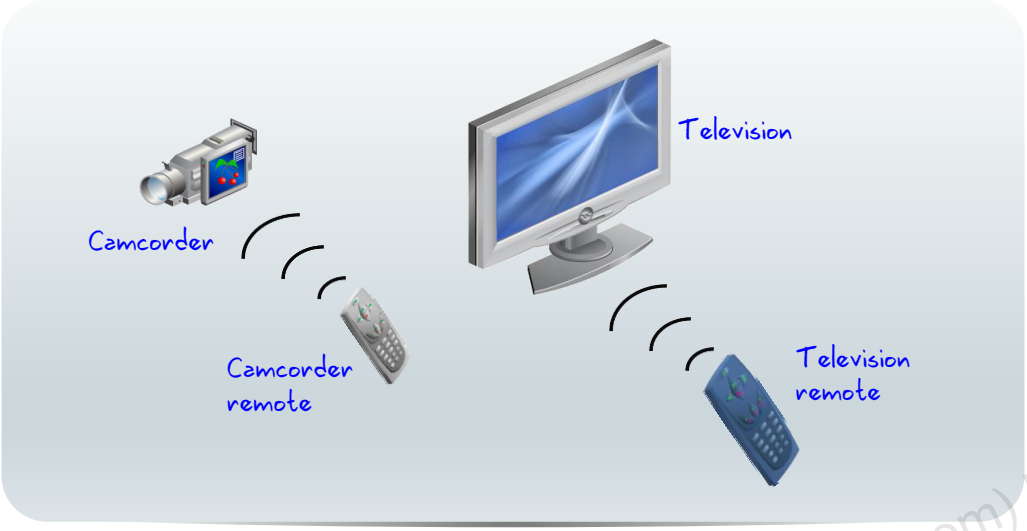


There is only one Camera object.

```
12  Camera remote1 = new Camera();
13
14  Camera remote2 = remote1;
15
16  remote1.play();
17
18  remote2.stop();
```


The diagram shows another important aspect of how references work. In this example, a `Camera` object is created and its reference assigned to a Camera reference, `remote1`. This reference is then assigned to another Camera reference, `remote2`. Now both references are associated with the same Camera object, and methods called on either reference will affect the same Camera object. Calling `remote1.play` is no different than calling `remote2.play`. Both remotes operate the same camera.

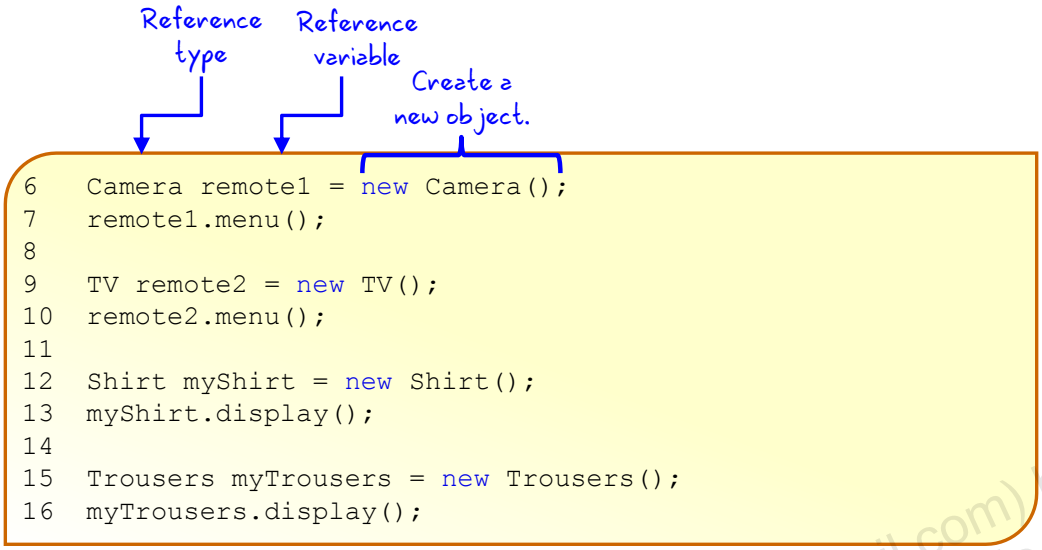
References to Different Objects



To extend the analogy just a little further, to work with a different type of object (for example, a television), you need a remote for that object. In the Java world, you need a reference of the correct type for the object that you are referencing.

You can ignore the fact that there is such a thing as a universal remote controller, although later in the course you will discover that Java also has the concept of references that are not limited to a single object type! For the moment, let's just say that a reference of the same type as the object is one of the reference types that can be used, and is a good place to start exploring the world of Java objects.

References to Different Objects



`remote1` references a `Camera` object.

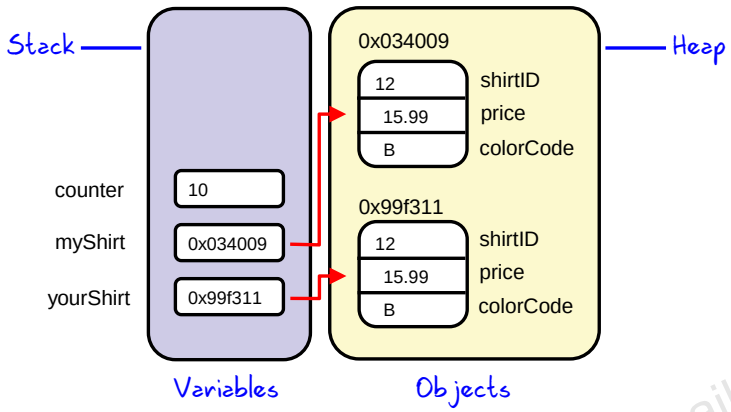
`remote2` references a `TV` object.

`myShirt` references a `Shirt` object.

`myTrousers` references a `Trousers` object.

References and Objects in Memory

```
12 int counter = 10;
13 Shirt myShirt = new Shirt();
14 Shirt yourShirt = new Shirt();
```

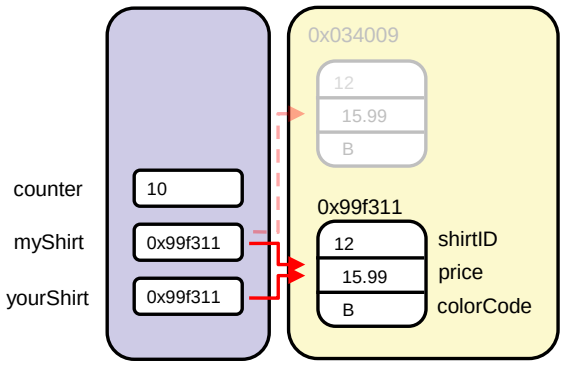


This diagram shows how references point to a particular object in memory. Note that there are two objects in memory, although they are both of type `Shirt`. Also note that there are two `Shirt` references pointing to these two `Shirt` objects.

The diagram also shows two types of memory that Java uses: the stack and the heap. The stack holds local variables, either primitives or reference types, whereas the heap holds objects. Later in this course, you will learn a little more about local variables, but for now it is sufficient to know that local variables are not fields of an object.

Assigning a Reference to Another Reference

```
myShirt = yourShirt;
```



The diagram shows what happens if the `myShirt` reference, after having its own object (in the previous slide), is now assigned the reference `yourShirt`. When this happens, the `myShirt` reference will drop its current object and be reassigned to the same object that `yourShirt` has. As a result, two references, `myShirt` and `yourShirt`, now point to the same object. Any changes to the object made by using one reference can be accessed using the other reference, and vice versa.

Another effect of assigning the reference `yourShirt` to the reference `myShirt` is that if the previous object referred to by `myShirt` has no other references, it will now be inaccessible. In due course, it will be garbage collected, meaning that its memory will become available to store other objects.

Two References, One Object

Code fragment:

```
12 Shirt myShirt = new Shirt();
13 Shirt yourShirt = new Shirt();
14
15 myShirt = yourShirt;    //The old myShirt object is
16                         //no longer referenced
17 myShirt.colorCode = 'R';
18 yourShirt.colorCode = 'G';
19
20 System.out.println("Shirt color: " + myShirt.colorCode);
```

Output from code fragment:

```
Shirt color: G
```

This example now shows what happens if you use either reference to make a change or get a value from the object. References `yourShirt` and `myShirt` refer to the same object, so making a change or getting a field value by using one reference is exactly the same as doing it with the other reference. The old object that was previously referenced by `myShirt` goes away.

Exercise 6-2: Modifying the `ShoppingCart` to Use `Item` Fields

1. Continue editing **`Exercise_06-1`** or open **`Exercise_06-2`** in NetBeans.
2. Create a new Java Main Class called `ShoppingCart`. This class contains a single main method. The rest of this exercise is spent modifying `ShoppingCart.java`.
3. Declare and instantiate two objects of type Item. Initialize only the descry field in each, using different values for each.
4. Print the description for each item and run the code.
5. (Optional) Above the code that prints the descriptions, assign item2 to item1. Run it again.

In this exercise, you declare and instantiate two variables of type Item in the `ShoppingCart` class and experiment with accessing properties and calling methods on the object.

Topics

- Describing objects and classes
- Defining fields and methods
- Declaring, instantiating, and using objects
- Working with object references
- Doing more with arrays
- Introducing the soccer league use case

Arrays Are Objects

Arrays are handled by an implicit Array *object*.

- The Array variable is an *object reference*, not a primitive data type.
- It must be instantiated, just like other objects.
  - Example:

```
int[] ages = new int[4];
```

This array can hold four elements.

- Previously, you have been using a shortcut to instantiate your arrays.
  - Example:

```
int[] ages = {8,7,4,5};
```

When you declare an array type variable, Java implicitly creates a special array class. Therefore, like other object types (`String` is an exception) it must be instantiated using the new keyword.

- In the top example, an `int` array called `ages` is declared and instantiated with a capacity to hold four elements.

Declaring, Instantiating, and Initializing Arrays

- Examples:

```
1    String[] names = {"Mary","Bob","Carlos"};
2
3    int[] ages = new int[3];
4    ages[0] = 19;
5    ages[1] = 42;
6    ages[2] = 92;
```

This statement declares an integer array ages and initializes the elements with default value of 0

- Not permitted (compiler will show an error):

```
int[] ages;
ages = {19, 42, 92};
```

As introduced in the lesson titled “Managing Multiple Items,” there are two approaches for creating and initializing arrays. Using the ***new*** keyword declares an array and initializes the elements of the array with default values based on the data type of the array. Default value for:

`int, short, byte and long` is 0

float and double is 0.0

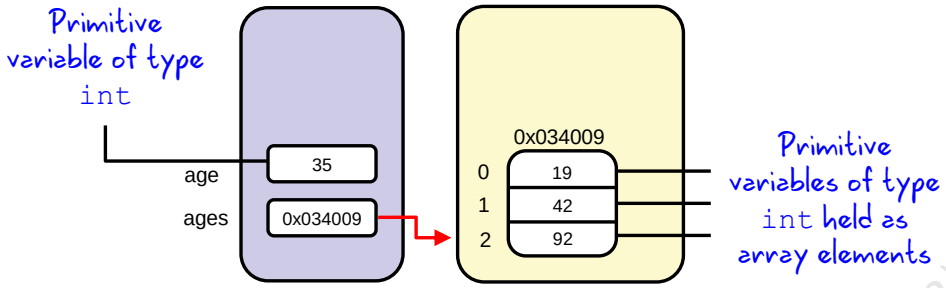
boolean to false

char with null character '\u0000'

String to null

Storing Arrays in Memory

```
int age = 35;
int[] ages = {19, 42, 92};
```



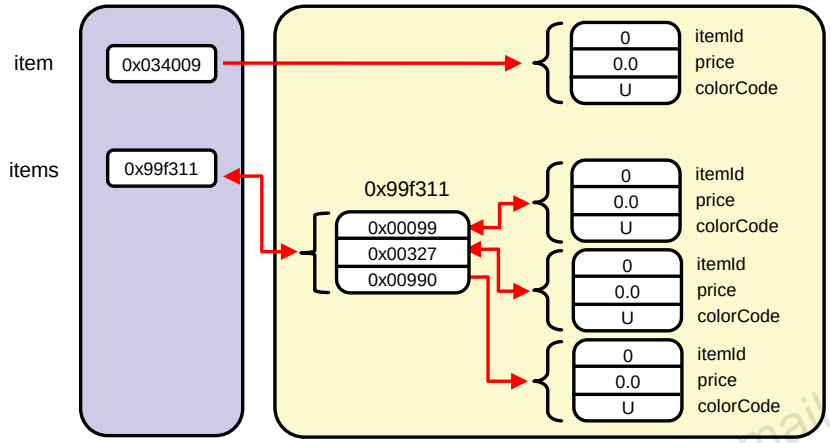
Arrays are objects referred to by an object reference variable. The diagram in the slide illustrates how a primitive array is stored in memory in comparison to how a primitive data type is stored in memory.

The value of the `age` variable (an int primitive) is `35`. The value of `ages` is `0x034009`, an object reference pointing to an object of type `array` (of `int` types) with three elements.

- The value of `ages[0]` is `19`.
- The value of `ages[1]` is `42`.
- The value of `ages[2]` is `92`.

Storing Arrays of Object References in Memory

```
Item item = new Item();
Item[] items = { new Item(), new Item(), new Item() };
```



The diagram in the slide illustrates how an object reference array is stored in memory. The value of the `item` object reference is x034009, which is an address to an object of type Item with the values 0, 0.0, and U.

The value of the `items[ ]` object reference is x99f311, which is an address to an object of type `Array` (of Item object references) containing three object references:

- The value of the `items[0]` index is 0x00099, which is an object reference pointing to an object of type Item.
- The value of the `items[1]` index is 0x00327, which is an object reference pointing to another object of type Item.
- The value of the `items[2]` index is 0x00990, which is an object reference pointing to another object of type Item.

Quiz

The following code is the correct syntax for _____ an array:

```
array_identifier = new type[length];
```

a. Declaring
b. Setting array values for
c. Instantiating
d. Declaring, instantiating, and setting array values for

Answer: c

a is incorrect. Declaring the array would look like this, assuming an array of object types: `Type[] array_identifier;`

b is incorrect. Setting array values would look like this, assuming an array of object types:

```
array_identifier[0]= new Type();
```

c is correct. The code example shows the array being initialized to a specific size.

d is incorrect. Declaring, instantiating, and setting array values would look like this, assuming an array of object types:

```
Type[] array_identifier = {new Type(), new Type(), new Type()};
```

Quiz

Given the following array declaration, which of the following statements are true?

```
int[ ] ages = new int[13];
```

a. `ages[0]` is the reference to the first element in the array.

b. `ages[13]` is the reference to the last element in the array.

c. There are 13 integers in the `ages` array.

d. `ages[5]` has a value of 0.

Answer: a, c, d

Topics

- Describing objects and classes
- Defining fields and methods
- Declaring, instantiating, and using objects
- Working with object references
- Doing more with arrays
- Introducing the soccer league use case

Soccer Application

Practices 6 through 14 build a soccer league application with the following features:

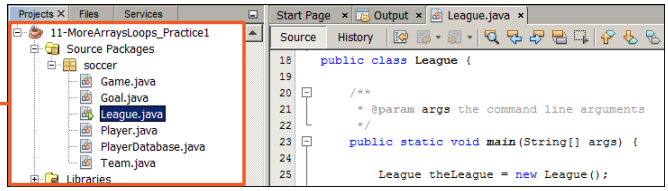
- Any number of soccer teams, each with up to 11 players
- Set up an all-play-all league.
- Use a random play game generator to create test games.
- Determine the rank order of teams at the end of the season.

In the remaining practices in this course, you will build an application that manages a Soccer League. The application will keep details on teams and players, as well as the results of games.

You will also write code that will randomly generate game results so that you can then develop code to list the Teams in rank order.

Creating the Soccer Application

A separate project for each practice



Sample output showing events in a game

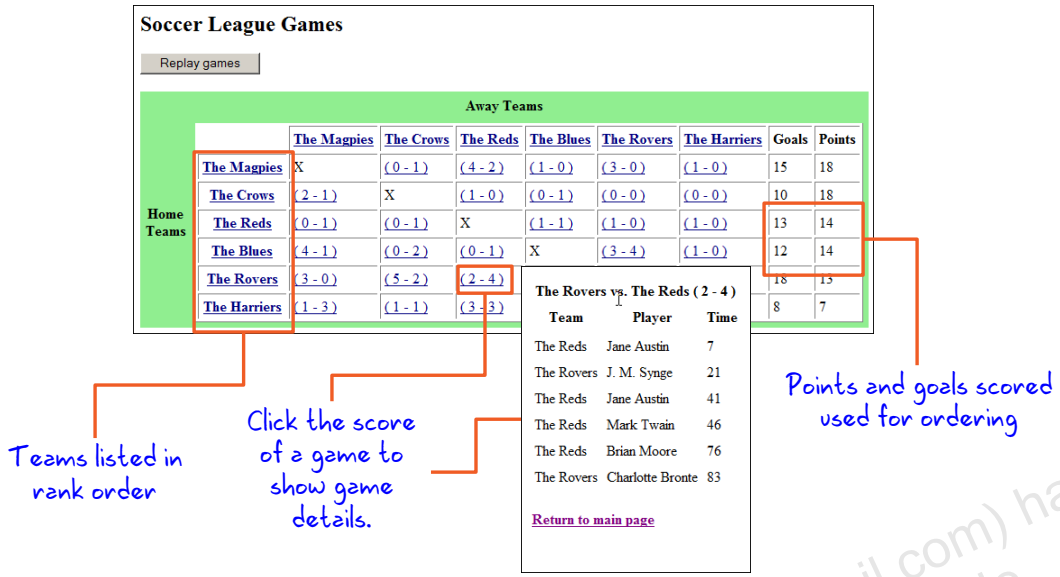
```
The Greys vs. The Pinks (2014-03-08)
Kickoff by Agatha Christie of The Greys. (0.0 mins.)
Arthur Conan Doyle of The Pinks currently has possession. (6.0 mins.)
GOAL! Scored by W. B. Yeats of The Greys. (7.0 mins.)
Kickoff by Alan Patton of The Pinks. (8.0 mins.)
Alexander Solzhenitsyn of The Pinks currently has possession. (11.0 mins.)
GOAL! Scored by Arthur Conan Doyle of The Pinks. (14.0 mins.)
Kickoff by Agatha Christie of The Greys. (18.0 mins.)
Alan Patton of The Pinks currently has possession. (23.0 mins.)
Agatha Christie of The Greys currently has possession. (24.0 mins.)
GOAL! Scored by Agatha Christie of The Greys. (40.0 mins.)
Kickoff by Arthur Conan Doyle of The Pinks. (44.0 mins.)
Arthur Conan Doyle of The Pinks currently has possession. (49.0 mins.)
GOAL! Scored by Arthur Conan Doyle of The Pinks. (55.0 mins.)
Kickoff by Agatha Christie of The Greys. (59.0 mins.)
Alan Patton of The Pinks currently has possession. (73.0 mins.)
GOAL! Scored by W. B. Yeats of The Greys. (89.0 mins.)
The Pinks win! (3 - 2)
```

Sample output showing rank order of teams

```
Team Points
The Reds:17:20
The Blues:17:17
The Pinks:12:17
The Greens:8:12
The Greys:6:13
BUILD SUCCESSFUL (total time: 0 seconds)
```


Initially, your application will be developed in NetBeans and you will see the results of running your code as text in the output window.

Soccer Web Application



The code that you write in the practices can be used by a simple web application to view the results of games in the league. You will see a demonstration of this.

Summary

In this lesson, you should have learned how to:

- Describe the characteristics of a class
- Define an object as an instance of a class
- Instantiate an object and access its fields and methods
- Describe how objects are stored in memory
- Instantiate an array of objects
- Describe how an array of objects is stored in memory
- Declare an object as a field

Practices Overview

- 6-1: Creating Classes for the Soccer League
- 6-2: Creating a Soccer Game