ORACLE
Academy

Java Programming

1-1

Fundamentals of Java - What I should know

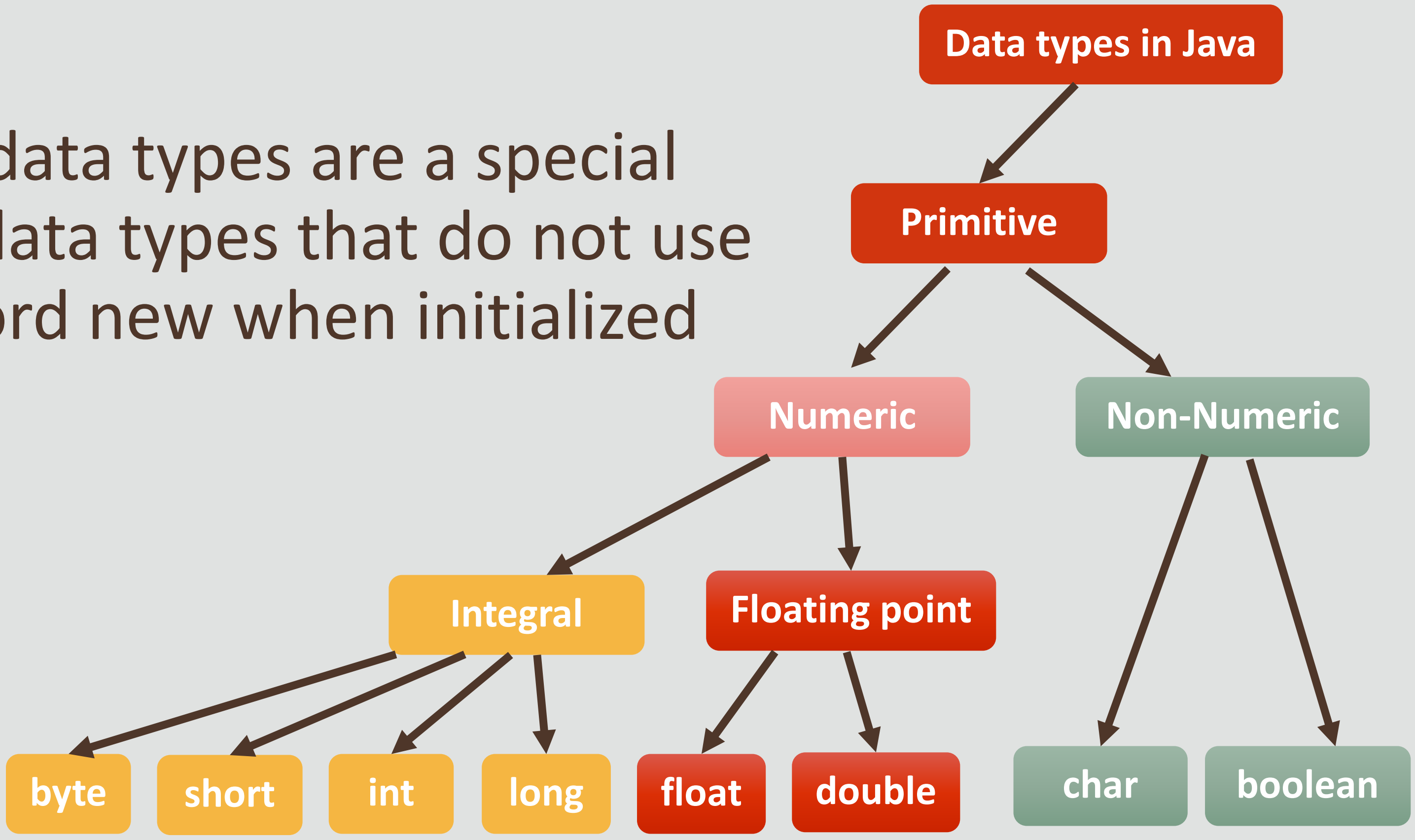
ORACLE
Academy

Objectives

- This lesson reviews the following topics:
  - Java Primitives
  - Strings
  - Logical and Relational Operators
  - Conditional Statements
  - Program Control
  - Object Classes
  - Constructor and Method Overloading
  - Inheritance

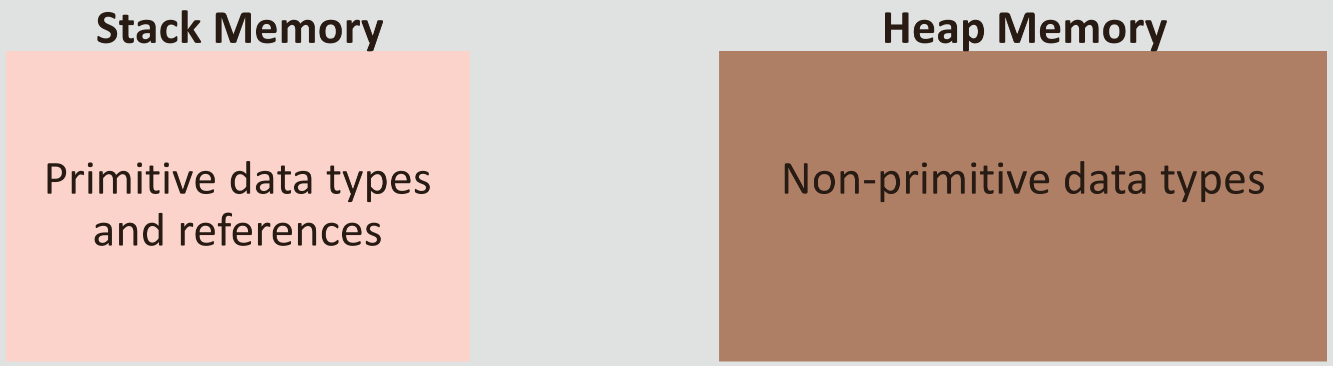
Primitive Data Types

- Java has eight primitive data types that are used to store data during a program's operation
- Primitive data types are a special group of data types that do not use the keyword new when initialized



Primitive Data Types

- Java creates primitives as automatic variables that are not references and are stored in stack memory with the name of the variable



- The most common primitive types used in this course are **int** (integers) and **double** (decimals)

Primitive Data Types

Data Type	Size	Example Data	Data Description
boolean	undefined (represents 1 bit of information)	true, false	true, false
byte	1 byte (8 bits)	12, 127	Stores integers from -128 to 127
char	2 bytes	'A', '5', '#'	Stores a 16-bit Unicode character
short	2 bytes	6, -14, 2345	Stores integers from -32,768 to 32,767.

Primitive Data Types

Data Type	Size	Example Data	Data Description
int	4 bytes	6, -14, 2345	Stores integers from: -2,147,483,648 to 2,147,483,647
long	8 bytes	3459111, 2	Stores integers from: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	3.145, .077	Stores a positive or negative decimal number from: 1.4023×10^{-45} to $3.4028 \times 10^{+38}$
double	8 bytes	.0000456, 3.7	Stores a positive or negative decimal number from: 4.9406×10^{-324} to $1.7977 \times 10^{+308}$

Declaring Variables and Using Literals

- The keyword new is not used when initializing a variable primitive type
- Instead, a literal value should be assigned to each variable upon initialization
- A literal can be any number, text, or other information that represents a value
- Examples of declaring a variable and assigning it a literal value:

```
boolean result = true;
char capitalC = 'C';
byte b = 100;
short s = 10000;
int i = 100000;
long creditCardNumber = 1234_5678_9012_3456L;
```

chars use single quotes, strings use double quotes.

Declaring Variables and Using Literals Task

1. Create a class named VarTest
2. Create the following 3 uninitialized variables:
   - int iNum;
   - char cVal;
   - boolean bVal;
3. Display all the values to the console
4. Update your code to assign the following:
   - iNum - 25
   - char cVal - B
   - boolean bVal - true
5. Display all the values to the console

Complete the task before checking your answer on the next slide!

ORACLE Academy

Variables and Literals Suggested Solution

```
package vartest;
public class VarTest {

   public static void main(String[] args) {
      int iNum = 25;
      char cVal = 'B';
      boolean bVal = true;

      //display all values to screen
      System.out.println(iNum);
      System.out.println(cVal);
      System.out.println(bVal);

   }//end method main

}//end class
```

Strings

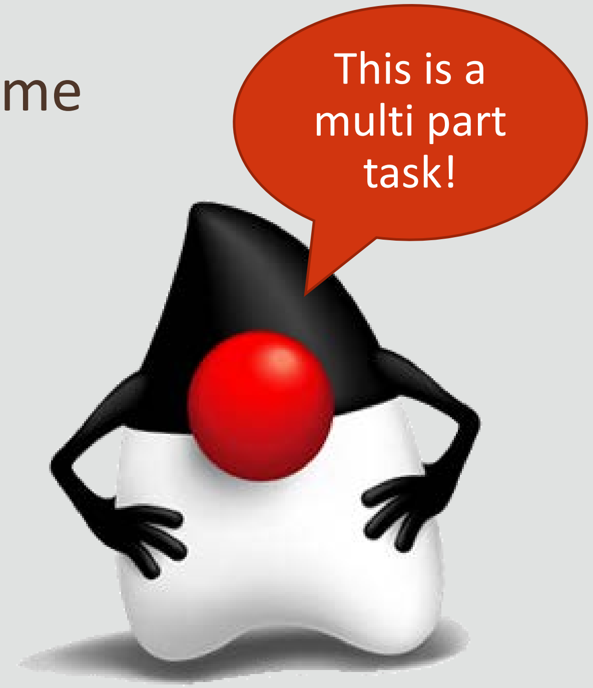
- A String is an object that contains a sequence of characters
- Declaring and instantiating a String is much like any other object variable
- However, there are differences:
  - They can be instantiated without using the new keyword.
  - They are immutable
  - Once instantiated, they are final and cannot be changed
- String objects come with their own set of methods

Remember that we should not use == to compare two strings.
This tests for reference equality and not value equality

String Operations Task

Instructions

1. Create a class named StringOperations
2. Create 3 string variables (str1, str2, str3)
   - str1 should be initialized to “Hello”
   - Str2 should be initialzed with your first name
   - Str3 should not be initialized with a value



String Operations Suggested Solution

```
package stringoperations;

public class StringOperations {

   public static void main(String[] args){

      String str1 = "Hello";
      String str2 = "Duke";
      String str3; //uninitialized string

   }//end method main

}//end class StringOperations
```

String Operations Task Continued

3. Assign a value to str3 that joins the literal value “You are “ to the name held in str2
4. Display a welcome message to screen that states “Welcome: “ and the value held in str3
5. Display the length of str1
6. Display a substring of str3 beginning with character 0, up to, but not including character 5
7. Display str2 in uppercase

ORACLE Academy

String Operations Suggested Solution

```
        //create a new string by concatenation
        str3 = "You are " + str2;
        //display a welcome message to screen
        System.out.println("Welcome: " + str3);
        //Display the length of a string
        System.out.println("Length: "+ str1.length());
        //Display a substring of str1 beginning with character 0,
        //up to, but not including character 5
        System.out.println("Sub: "+ str3.substring(0,5));
        //display a string value in uppercase
        System.out.println("Upper: "+str2.toUpperCase());

    }//end method main

}//end class StringOperations
```

String Operations

- You should never compare the value of Strings by using the == method (equals equals) that you use with primitive data types

str1 == str2

- This is because the == operator only compares the contents of stack memory therefore only compares the String references not the String values

String Operations

- When comparing Strings there are different methods that you can use.
- The two ways of doing this are the compareTo() method and the equals() method
- Your options are:
  - compareTo(String str)
  - compareToIgnoreCase(String str)

  or
  - Equals(Object obj)
  - EqualsIgnoreCase(String str)

String Operations

- Method: s1.compareTo(s2);
- Should be used when trying to find the lexicographical order of two strings
- Returns an integer
  - If s1 is less than s2, an int < 0 is returned
  - If s1 is equal to s2, 0 is returned
  - If s1 is larger than s2, an int > 0 is returned
- There is also an option that allows you to ignore the case of the Strings

String Operations

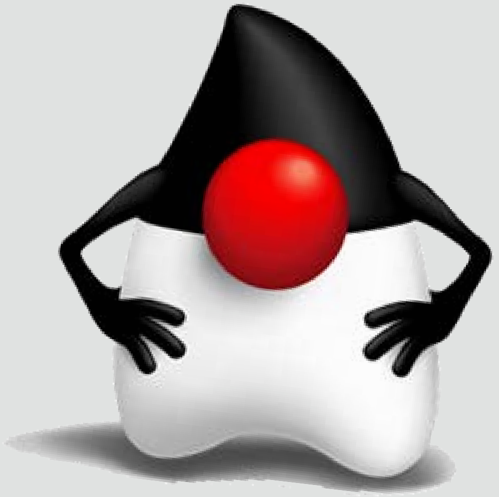
- Method: s1.equals(s2)
- Should be used when you only wish to find if the two strings are equal
- Returns a boolean value
  - If true is returned, s1 is equal to s2
  - If false is returned, s1 is not equal to s2

This will only ever test for equality, not order!

Academy

String Operations Task Continued

8. Use a comparison method to determine if the value of str1 and str2 are the same
   - If they are not, display the information that gives you the lexicographical order of the two Strings
9. Display the value that tells us whether two Strings are the same
   - The order is not important



String Operations Suggested Solution

```
        //Compare 2 Strings to see if they are the same or
        //identify which String value comes first
        System.out.println(str1.compareTo(str2));
       //Compare 2 Strings to see if they are the same
        System.out.println(str1.equals(str2));

    }//end method main

}//end class StringOperations
```

Console Input

- To read input that the user has entered to the console, use the Java class Scanner. You will have to use an import statement to access the class java.util.Scanner

```
import java.util.Scanner;
```

- To initialize a Scanner, write:

```
Scanner in = new Scanner(System.in);
```

You should explore the scanner class in more depth. You will see there are many methods for reading different types from the console

Console Input

- To accept different data types from the console use:
  - To read a single String value:
    ```
    String txtInput = in.next();
    ```
  - To read a line of text:
    ```
    String multiTxtInput = in.nextLine();
    ```
  - To read the next integer:
    ```
    int numValue = in.nextInt();
    ```
  - To read the next double:
    ```
    double doubleValue = in.nextDouble();
    ```

Console Input Task

1. Create a class named InputVariables
2. Create a Scanner object named in
3. Create an initialized variable for each primitive data type

Console Input Suggested Solution

```java
package inputvariables;
import java.util.Scanner;
public class InputVariables {

   public static void main(String[] args) {
      Scanner in = new Scanner(System.in);

      boolean boolVal;
      byte byteVal;
      char charVal;
      short shortVal;
      int intVal;
      long longVal;
      float floatVal;
      double doubleVal;
   }//end method main

}//end class InputVariables
```

Console Input Task

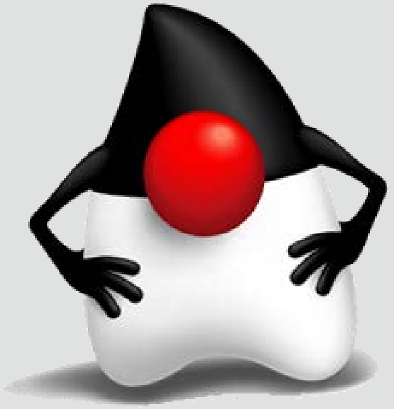
4. Prompt for and read in a value for each variable using the Scanner. Use a print statement to create a prompt that resembles the following:

```
Please enter a boolean value:
```

5. Display the values for each variable that were entered in the console by using a println statement

```
boolean value: true
```

Close the scanner when you no longer need it



Console Input Suggested Solution

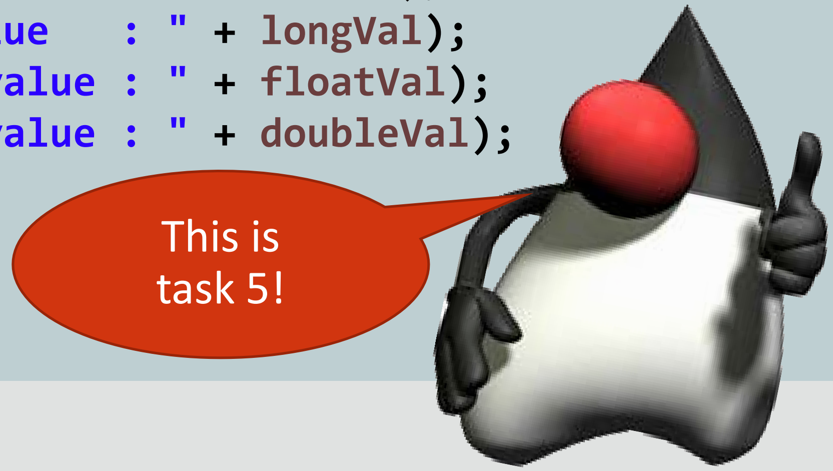
```
System.out.print("Please enter a boolean value: ");
boolVal = in.nextBoolean();

System.out.print("Please enter a byte value: ");
byteVal = in.nextByte();

System.out.print("Please enter a char value: ");
charVal = in.next().charAt(0);

System.out.print("Please enter a short value: ");
shortVal = in.nextShort();

System.out.print("Please enter an int value: ");
intVal = in.nextInt();

System.out.print("Please enter a long value: ");
longVal = in.nextLong();

System.out.print("Please enter a float value: ");
floatVal = in.nextFloat();
```

Continued on next slide ….

Console Input Suggested Solution

Continued from previous slide ….

```
        System.out.print("Please enter a double value: ");
        doubleVal = in.nextDouble();
        in.close();

        System.out.println("boolean value: " + boolVal);
        System.out.println("byte value    : " + byteVal);
        System.out.println("char value    : " + charVal);
        System.out.println("short value   : " + shortVal);
        System.out.println("int value     : " + intVal);
        System.out.println("long value    : " + longVal);
        System.out.println("double value  : " + floatVal);
        System.out.println("double value  : " + doubleVal);

    }//end method main

}//end class InputVariables
```



Relational Operators

- Java has six relational operators used to test primitive or literal numerical values
- Relational operators are used to evaluate if-else and loop conditions

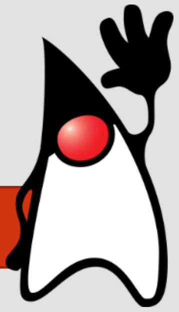
Relational Operator	Definition
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	Equal to
!=	Not equal to

Logic Operators

- Java has three logic operators used to combine boolean expressions into complex tests

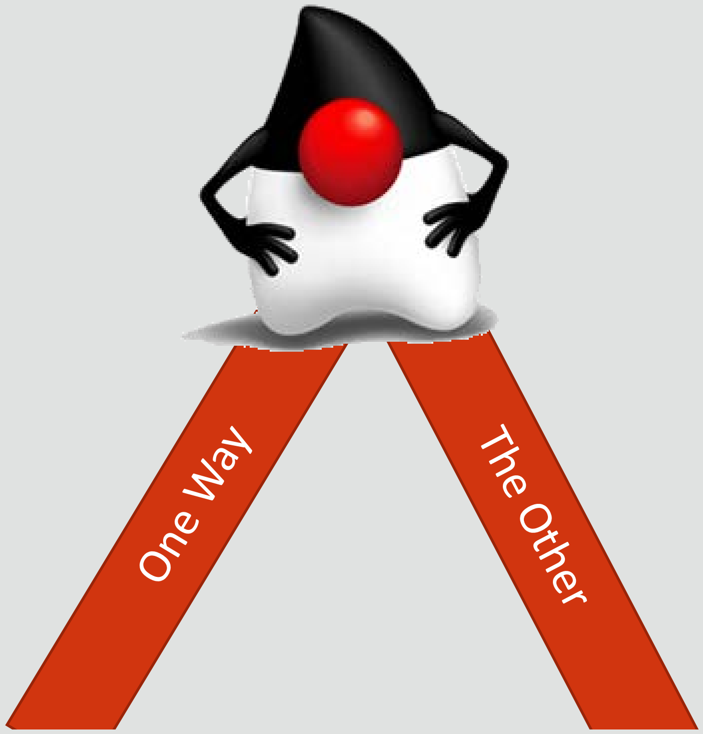
Logic Operator	Meaning
&&	And
||	Or
!	Not

Understanding boolean logic is fundamental to programming.



If Conditional Statement

- To build an if-else statement, remember the following rules:
  - An if-else statement needs a boolean condition or method that is tested for true/false
- For example:
  - if(x==5)
  - if(y >= 17)
  - if(s1.equals(s2))



Conditional Statement Task

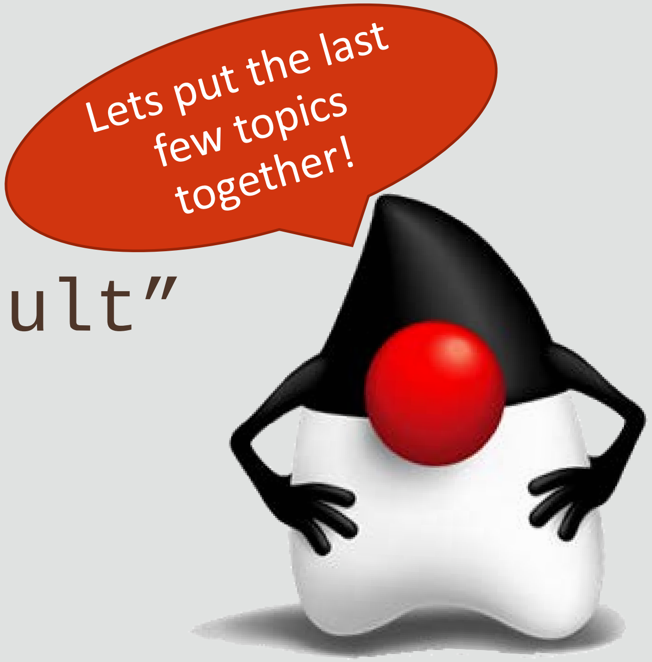
1. Create a class named AgeChecker
2. Read in an integer value for the users age from the console
3. Combine an if/else statement with a greater than relational operator to display

```
        “You are an adult”
```

   if the age is 21 or over, or

```
        “You are not an adult”
```

   if the age is less than 21



ORACLE Academy

Conditional Statement Suggested Solution

```
package agechecker;
import java.util.Scanner;
public class AgeChecker {

   public static void main(String[] args) {
      Scanner in = new Scanner(System.in);
      int age;

      System.out.print("Please enter your age: ");
      age = in.nextInt();

      if(age > 20)
         System.out.println("You are an adult");
      else
         System.out.println("You are not an adult");
      //endif
      in.close();
   }//end method main

}//end class AgeChecker
```

Conditional Statement Task

- You need to be both 21 or over and have a driving licence to hire a car

4. Read in a character value to identify if the user holds a current driving licence
   - Accept ‘y’ for yes and ‘n’ for no, assume that the input will be in the correct format

Conditional Statement Suggested Solution

```
public class AgeChecker {
   public static void main(String[] args) {
      Scanner in = new Scanner(System.in);
      int age;
      char holdLicence;

      System.out.print("Please enter your age: ");
      age = in.nextInt();
      System.out.print("Do you hold a current driving
                                  licence? ");
      holdLicence = in.next().charAt(0);

      if(age > 20)
         System.out.println("You are an adult");
      else
         System.out.println("You are not an adult");
      //endif
      in.close();
   }//end method main
}//end class AgeChecker
```

ORACLE Academy

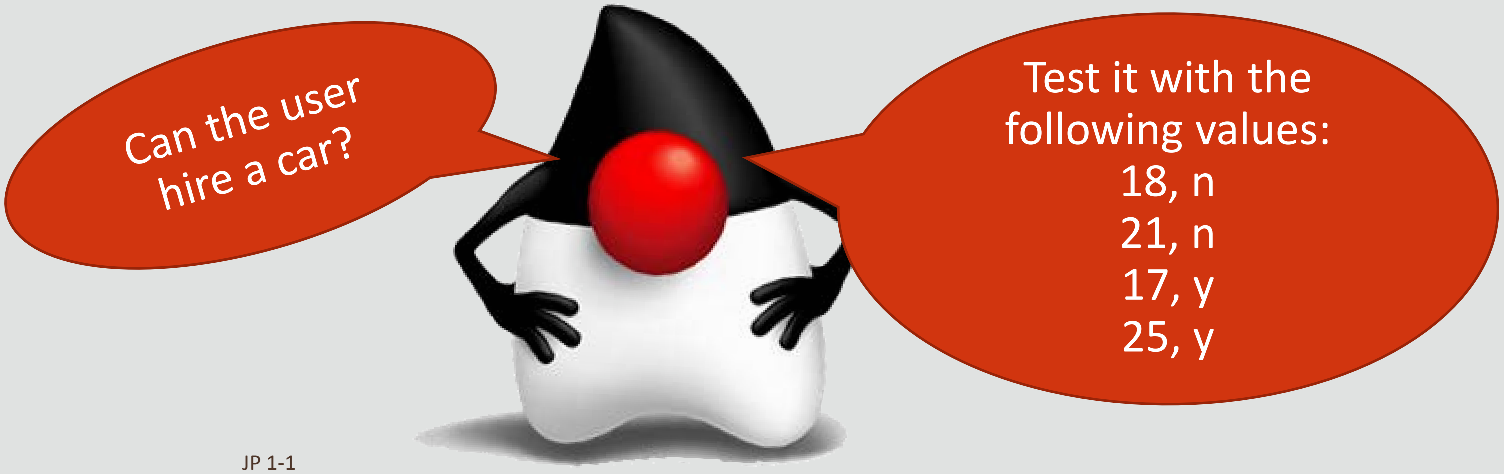
Conditional Statement Task

5. Add a logical operator to the if/else statement so that if you are 21 or over and have a driving licence you can hire a car, the output should be:

```
“You are an adult and can hire a car”
```

Or

```
“You are not an adult so cannot hire a car”
```



ORACLE Academy

Conditional Statement Suggested Solution

```
public class AgeChecker {
   public static void main(String[] args) {
      Scanner in = new Scanner(System.in);
      int age;
      char holdLicence;
      System.out.print("Please enter your age: ");
      age = in.nextInt();
      System.out.print("Do you hold a current driving
                                   licence? ");
      holdLicence = in.next().charAt(0);
      if((age > 20) && (holdLicence == 'y'))
         System.out.println("You are an adult and can
                                      hire a car");
      else
         System.out.println("You are not an adult so
                                      cannot hire a car");
      //endif
      in.close();
   }//end method main
}//end class AgeChecker
```

ORACLE Academy

Conditional Statement Task 2

1. Create a class called ValueChecker
2. Prompt for and read in an integer value
3. If the number is 7 then display “That’s lucky!”
4. If the number is 13 then display “That’s unlucky!”
5. If the number is anything else then display “That number is neither lucky nor unlucky!”

Conditional Statement Task 2 Suggested Solution

```
import java.util.Scanner;
public class ValueChecker {
  public static void main(String[] args){
    Scanner in = new Scanner(System.in);
    int value = 0;
    System.out.println("Enter a number:");
    value = in.nextInt();
    if( value == 7) {
      System.out.println("That's lucky!");
    }
    else if( value == 13) {
      System.out.println("That's unlucky!");
    }
    else {
      System.out.println("That number is neither lucky nor unlucky!");
    }//end if
    in.close();
  }//end method main
}//end class ValueChecker
```

Loops - While

- With a while loop, Java uses the syntax:

```
while(condition is true){
  //logic
}
```

- Similar to if statements, the while loop parameters can be boolean types or can equate to a boolean value
- Conditional statements (<, >, <=, >=, !=, ==) equate to boolean values.
  - Examples:
    - while (num1 < num2)
    - while (isTrue)
    - while (n !=0)

Loops - do-while

- With a do-while loop, Java uses the syntax:

```
do{
   //statements to repeat go here
} while(condition);
```

- The do-while loop:
  - Is a post-test loop
  - Is a modified while loop that allows the program to run through the loop once before testing the boolean condition.
  - Continues until the condition becomes false
- If you do not allow for a change in the condition, both the while and do-while loops will run forever as an infinite loop

Loops - for

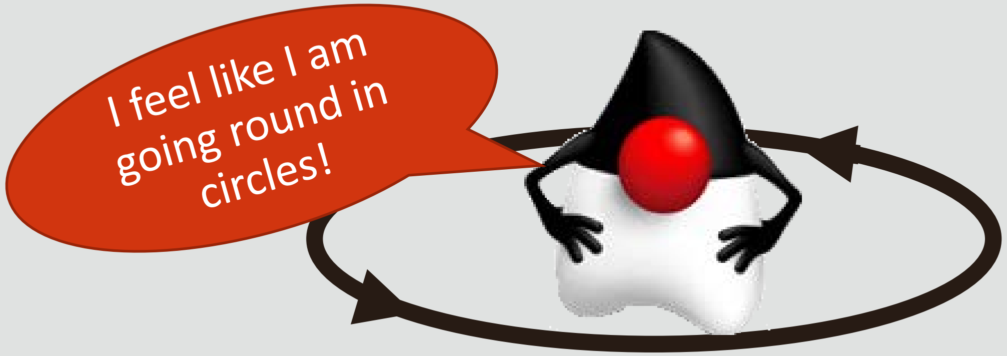
- With a for loop, Java uses the syntax:

```
for(int i=0; i < timesToRun; i++){
    //logic
}//endfor
```

- The for loop repeats code a pre set number of times
- The syntax of a for loop contains three parts:
  - Initializing the loop counter (**int i=0;**)
  - Conditional statement, or stopping condition (**i < timesToRun;**)
  - Updating the counter (going to the next value(**i++**))
    - Think of i as a counter starting at 0 and incrementing until the value of I fails the stopping condition

Loops Task

1. Open the ValueChecker class in your IDE
2. Use a do-while loop so the program will continually ask the user for lucky numbers and display the result to the console
3. The program should terminate when the user enters a 0 (zero) as the value
4. No message should be displayed if a 0 value is entered



Loops Suggested Solution

```
public class ValueChecker {
  public static void main(String[] args){
    Scanner in = new Scanner(System.in);
    int value = 0;
    do {
       System.out.println("Enter a number:");
       value = in.nextInt();
       if( value == 7)
         System.out.println("That's lucky!");
       else if( value == 13)
         System.out.println("That's unlucky!");
       else if(value!=0){
         System.out.println("That number is neither lucky nor unlucky!");
       //end if
    }while(value!=0);
    in.close();
  }//end method main

}//end class ValueChecker
```

Array

- An array is a collection of values of the same data type stored in a container object
- Length of the array is set when the array is declared.
- Array examples:

```
String[] myBouquet = new String[6];   // Length = 6

int[] numbers = {7, 24, 352, 2, 37};  // Length = 5
```

- The address elements range from 0 to length-1

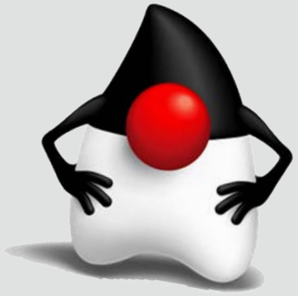
Address element	**0**	**1**	**2**	**3**	**4**
Value	7	24	352	2	37

ORACLE Academy

Array Task

- A program is required to store a range of unique values!

1. Create a class called UniqueNums
2. Create a 5 element integer array called numbers
3. Create the following 4 variables:
   - integer value called num, initialized to 0 that stores the users input.
   - Integer value called numValues, initialized to 0 that store the number of valid entries.
   - Boolean value called valid, initialized to true that flags whether the number is unique or not.
   - Scanner object called in.



Array Suggested Solution

```
import java.util.Scanner;

public class UniqueNums {

   public static void main(String[] args) {

      int[] numbers = new int[5];
      Scanner in = new Scanner(System.in);
      int num=0, numValues=0;
      boolean valid = true;

   }//end method main

}//end class UniqueNums
```

Array Task Continued

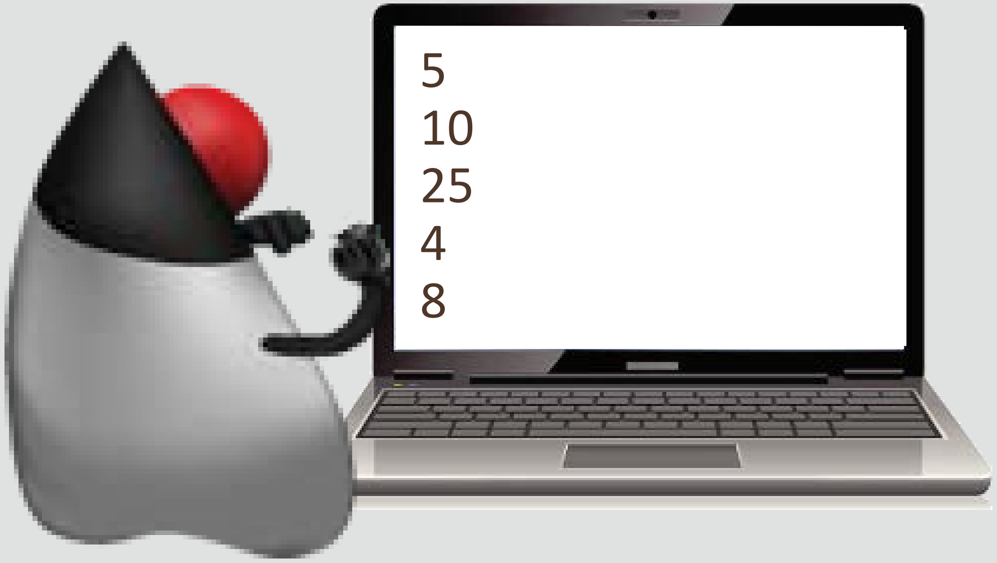
4. Use a while loop to create a control structure that will continually execute until the number of values entered exceeds the size of the array
5. Use a nested do while loop to ask the user for a valid unique value
6. Use a nested for loop to check that the entered value does not already exist in the numbers array
7. If a valid number has been entered add it to the numbers array
8. Use the numValues variable for the array index
9. Increment the numValues variable

Array Suggested Solution Continued

```
      while(numValues<numbers.length) {
         do {
            valid=true;
            System.out.print("Please enter a unique number: ");
            num = in.nextInt();
            for(int i = 0; i<numValues; i++)
            {
               if(num == numbers[i]) {
                  System.out.println("Number already exists");
                  valid = false;
                  break;
               }//endif
            }//endfor
         }while(!valid);
         numbers[numValues] = num;
         numValues++;
      }//endwhile
   }//end method main
}//end class UniqueNums
```

Array Task Continued

10. When the array has been filled with unique values close the Scanner object
11. Use an enhanced for loop to display the contents of the numbers array to the console



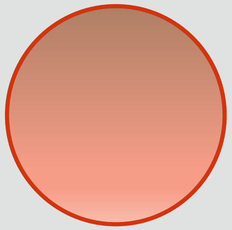
```
5
10
25
4
8
```

Array Suggested Solution Continued

```
        in.close();

        for(int numV: numbers)
           System.out.println("Number Value: " + numV);
        //endfor

     }//end method main

}//end class UniqueNums
```

Object Classes

- Are classes that define objects to be used in a driver class
- Can be found in the Java API, or created by you
- Examples:
  - Animal
  - Circle
  - String
  - Student



Student Class Example

```
package com.example.domain;
public class Student{
     private int studentId;
     private String name, ssn;
     private double gpa;
     private final int SCHCODE = 34958;
  public Student(){
  }//end constructor method
  public int getStudentId()
  {
    return studentId;
  }//end method getStudentId
  public void setStudentId(int x)
  {
     studentId = x;
  }//end method setStudentId
}//end class Student
```

Package Statement

Class Declaration

Instance Fields (Variables)

Constructor

Methods

Instance Fields

- An instance field is a variable declared within the class for which every object of the class will hold its own value
- Instance fields are declared within the class but outside of the methods

Class Declaration
Instance Fields
Methods

- Instance fields should be made private and accessed through a getter method

Access Modifiers

- Access modifiers specify accessibility to changing variables, methods, and classes
- There are four access modifiers in Java:

Access Modifier	**Description**
public	Allows access from anywhere.
protected	Allows access only inside the package containing the modifier.
private	Only allows access from inside the same class.
Default (not specified/blank)	Allows access inside the class, subclass, or other classes of the same package as the modifier.

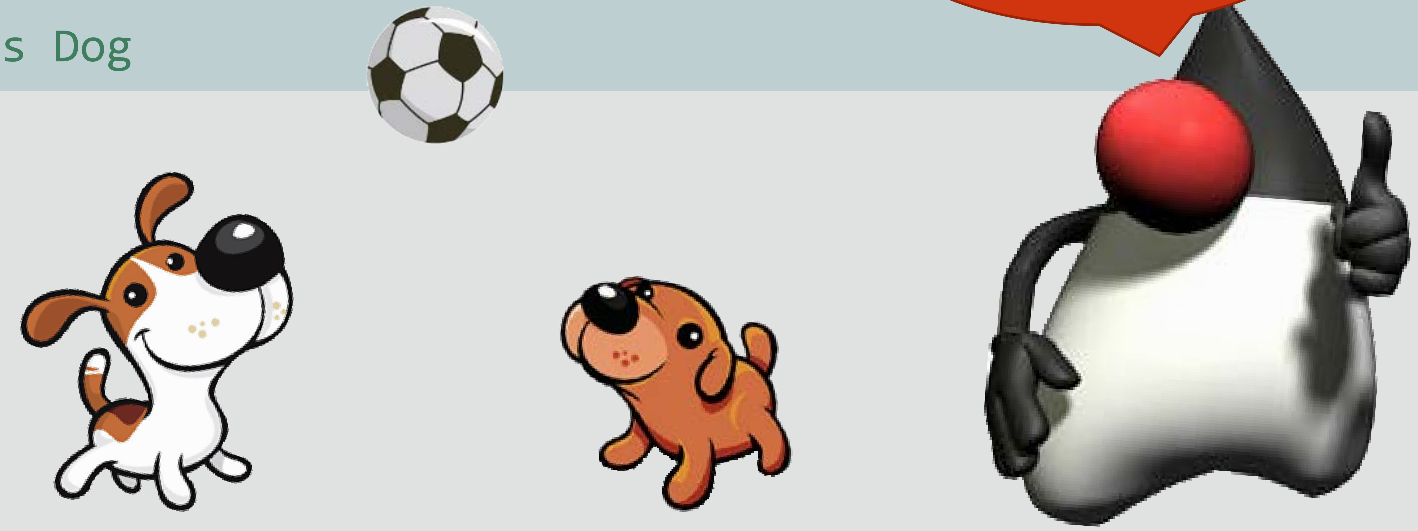
- Fields are typically private and methods public, but this is not always the case

Object Classes and Instance Fields Task

1. Create a project named animalshop
2. Create a class called Dog that does not have a main method
3. Include the following instance fields in the dog class:
   - String name
   - String breed
   - String barkNoise – “Woof”
   - Double weight

Object/Instance Fields Suggested Solution

```
package animalshop;

public class Dog {
   //instance field declarations
   private String name;
   private String breed;
   private String barkNoise = "Woof";
   private double weight;

}//end class Dog
```



Constructor

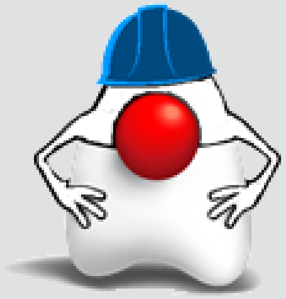
- A constructor is a method that creates an object
- In Java, constructors are methods with the same name as their class, used to create an instance of an object
- Constructors are invoked using the new keyword
- You can declare more than one constructor in a class declaration as long as they have different signatures

Constructor

- Example creating an object using Student constructor:

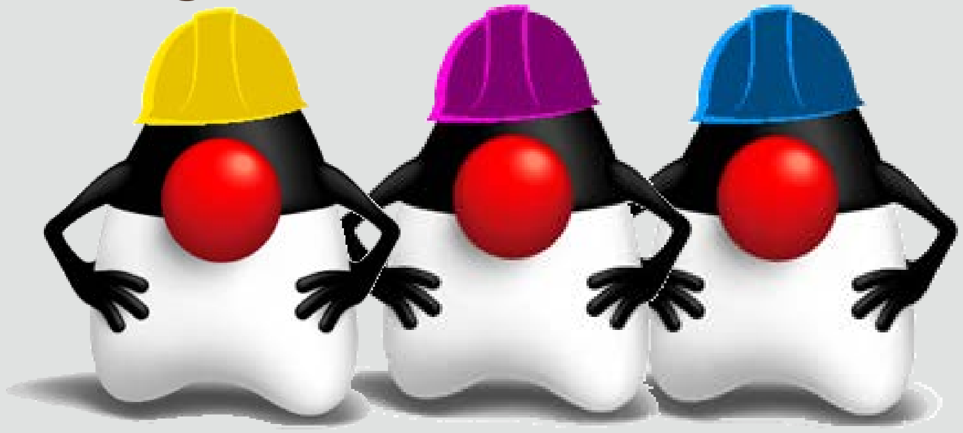
```
Student stu = new Student();
```

- The constructor call is at the end of the statement and must include arguments that match the prarmeter list of the constructor
- If you do not declare a constructor, Java will provide a default (blank) constructor for you
- When you declare your own constructor the default one provided by Java will no longer be available



Overloading Constructors

- Constructors assign initial values to instance variables of a class
- Constructors inside a class are declared like methods
- Overloading a constructor means having more than one constructor with the same name
- However the number of arguments would be different, and/or the data types of the arguments would differ



Constructor with Parameters

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

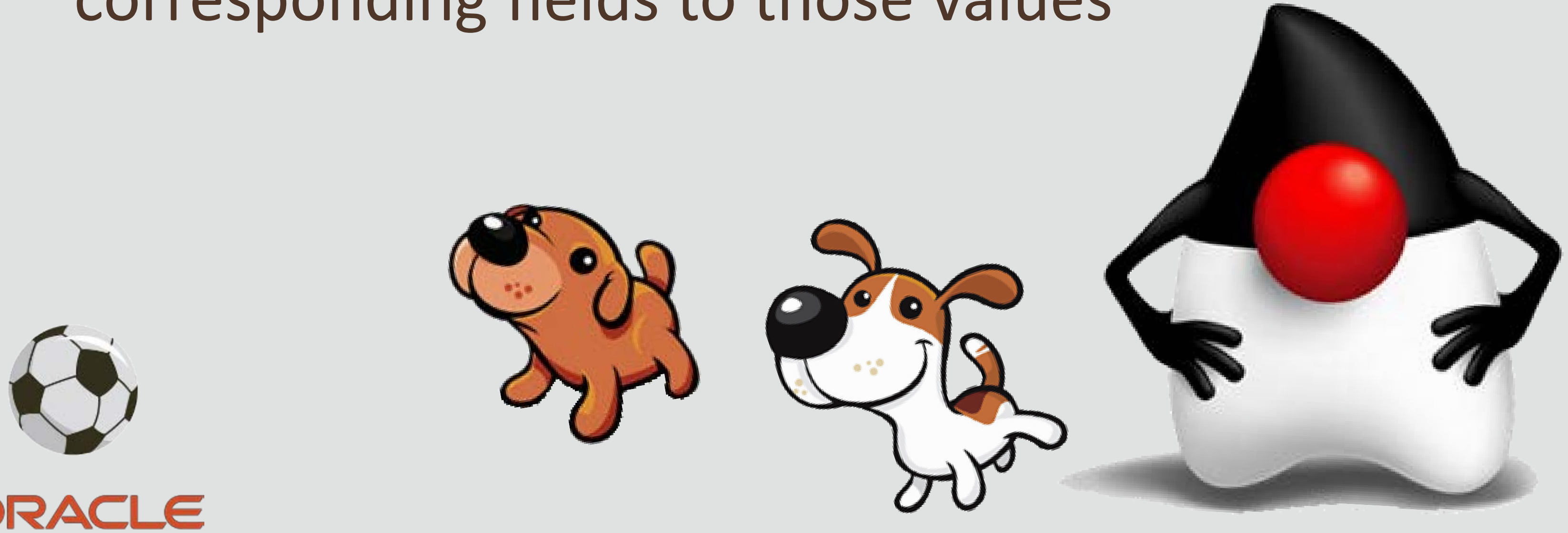
```
public Student(String n, String ssn) {
   name = n;
   this.ssn = ssn;
}//end class Student
```

- To instantiate a Student instance using the constructor with parameters, write:

```
Student student1 = new Student("Zina", "3003456");
```

Object/Instance Fields Task Continued

4. Create a constructor that creates a dog object using parameters for the name, breed and weight fields. Assign the instance fields to these values
5. Create a second constructor that accepts values for all instance fields as parameters and assigns the corresponding fields to those values



Object/Instance Fields Suggested Solution

```
    public Dog(String name, String breed, double weight) {
        this.name = name;
        this.breed = breed;
        this.weight = weight;
    }//end constructor method

    public Dog(String name, String breed, String noise, double weight) {
        this.name = name;
        this.breed = breed;
        barkNoise = noise;
        this.weight = weight;
    }//end constructor method

}//end class Dog
```

Components of a Method

- A method signature is the name and parameters.
- 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

```
public void displayName(){
   System.out.println(createName());
}//end method displayName

private String createName()
{
  return( this.firstName + " " + this.lastName );
}//end method getName
```

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

Return Type

Name of Method

Parameters

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

Class Methods

- Every class can have a set of methods associated with it which allow functionality for the class
  - Accessor method
    - Often called "getter" method
    - Returns the value of a specific instance variable
  - Mutator method
    - Often called "setter" method
    - Changes or sets the value of a specific instance 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 instance field of the class
- Non-void return type corresponds to the data type of the instance field you are accessing
- Includes a return statement
- Usually have no parameters

```
public String getName(){
  return name;
}//end method getName

public int getStudentId(){
  return studentId;
}//end method getStudentId
```

Mutator Methods

- Mutator methods set or modify the value of a specified instance field of the class
- Have a void return type
- Parameter has a type that corresponds to the type of the instance field being set

```
public void setName(String name){
   this.name = name;
}//end method setName

public void setStudentId(int id){
   studentId = id;
}//end method setStudentId
```

Functional Methods

- Functional methods perform a function or behavior for the class
- Can have either void or non-void return type
- Parameters are optional and used depending on what is needed for the method's function

```
private String createName()
{
  return( this.firstName + " " + this.lastName );
}//end method getName
```

ORACLE
Academy

Object/Instance Fields Task Continued

6. Create Accessor and Mutator (getter and setter) methods for the following fields:
   - name
   - breed
   - Weight
7. **Do not** create one for the barkNoise field!

Object/Instance Fields Suggested Solution

```
    public String getName() {
       return name;
    }//end method getName

    public void setName(String name) {
       this.name = name;
    }//end method setName

    public String getBreed() {
       return breed;
    }//end method getBreed

    public void setBreed(String breed) {
       this.breed = breed;
    }//end method setBreed
```

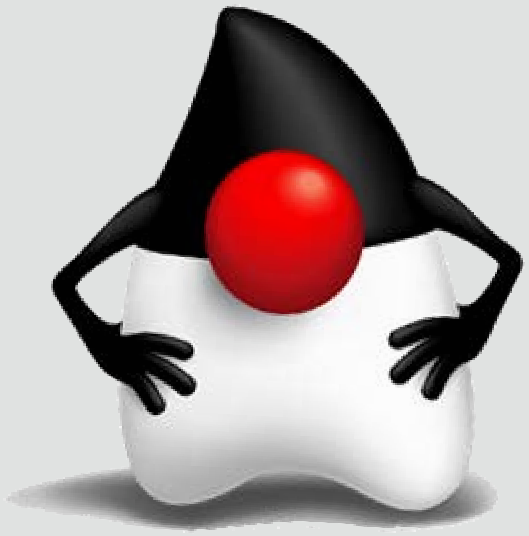
Continued on next slide ....

Object/Instance Fields Suggested Solution

```
    public double getWeight() {
       return weight;
    }//end method getWeight

    public void setWeight(double weight) {
       this.weight = weight;
    }//end method setWeight

 }//end class Dog
```

Object/Instance Fields Task Continued

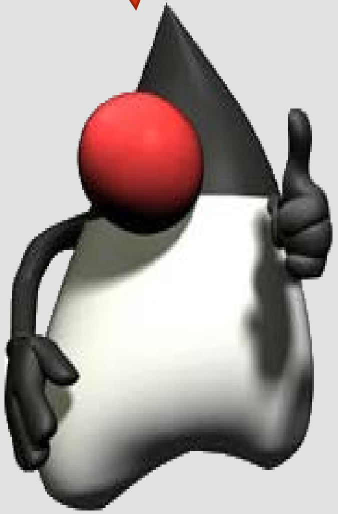
8. Add a functional method called bark that will display the value of the barkNoise field to the console



Object/Instance Fields Suggested Solution

```
    public void bark(){
       System.out.println(barkNoise);
    }//end method bark

}//end class Dog
```

Now you know what your dogs are saying!!

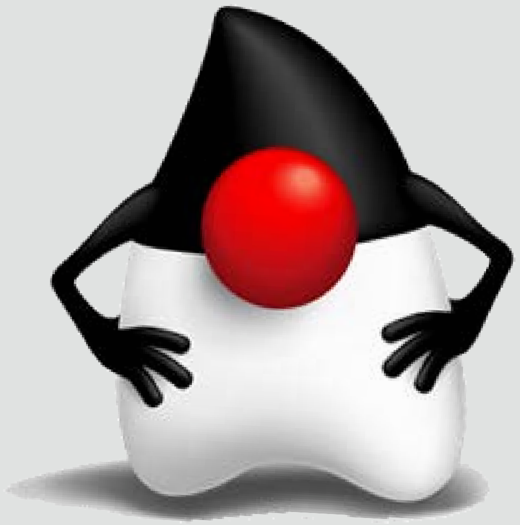


Overloading Methods

- Just as you used overloading when you created your constructors you can also overload methods (behaviours)
- An overloaded method must have a different method signature
  - The number of parameters must be different
  - The type of parameters must be different
  - The number or order of the parameter types must be different
- The name of the parameters are not part of the method signature

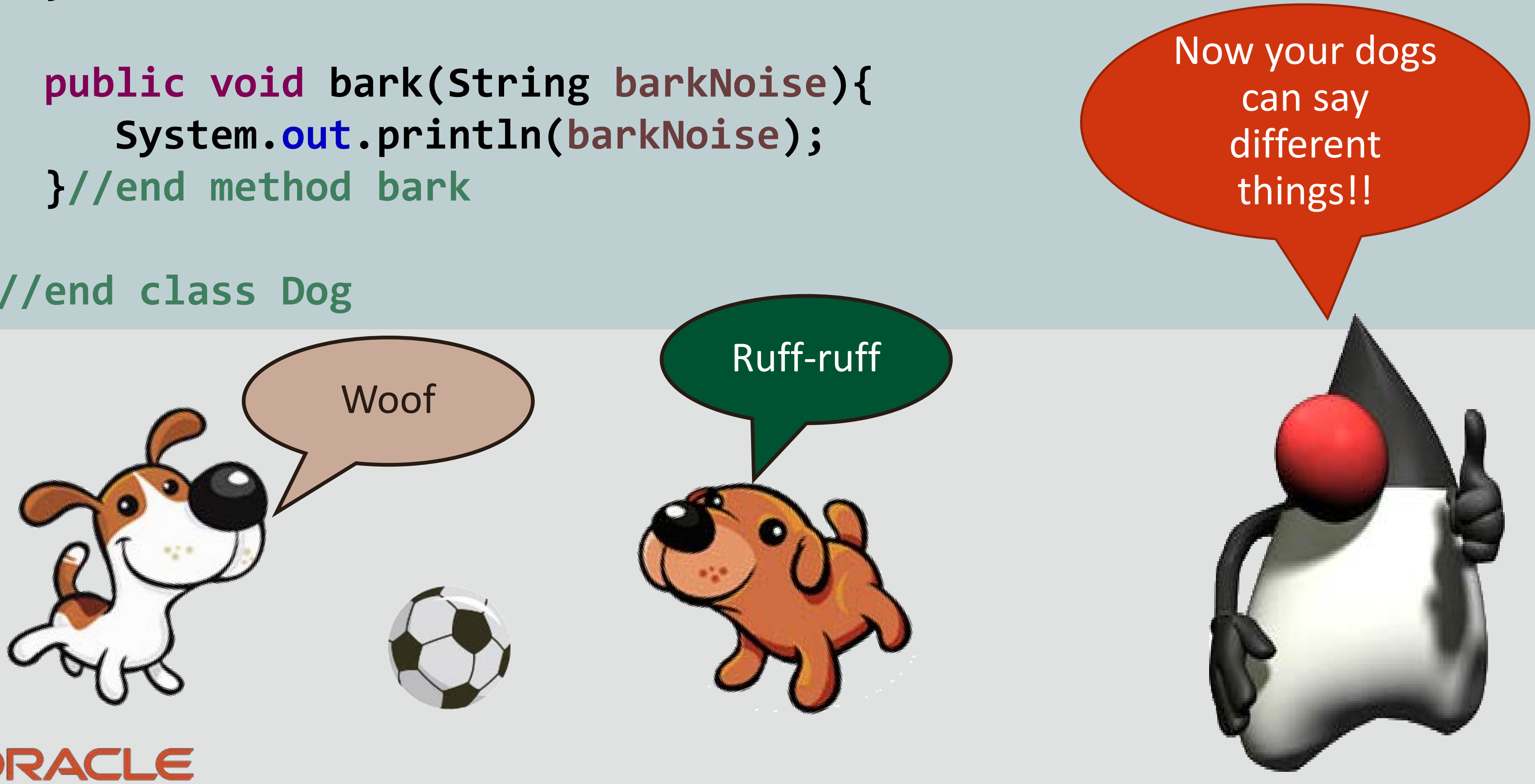
Object/Instance Fields Task Continued

9. Add an overloaded functional method called bark that will accept a String parameter for the noise the dog will make when it barks



Object/Instance Fields Suggested Solution

```
    public void bark(){
        System.out.println(barkNoise);
    }//end method bark

    public void bark(String barkNoise){
        System.out.println(barkNoise);
    }//end method bark

}//end class Dog
```



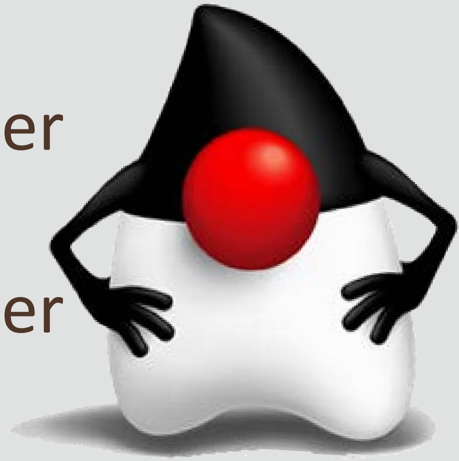
Main Method

- To run a Java program you must define a main method in a Driver Class
- The main method is automatically called when the class is called
- Example:

```
public class StudentTester{

   public static void main(String args[])
   {

   }//end method main

}//end class StudentTester
```

Object/Instance Fields Task Continued

10. Create a class called AnimalTester that contains a main method
11. Create a dog1 object using the 3 parameter constructor
12. Create a dog2 object using the 4 parameter constructor

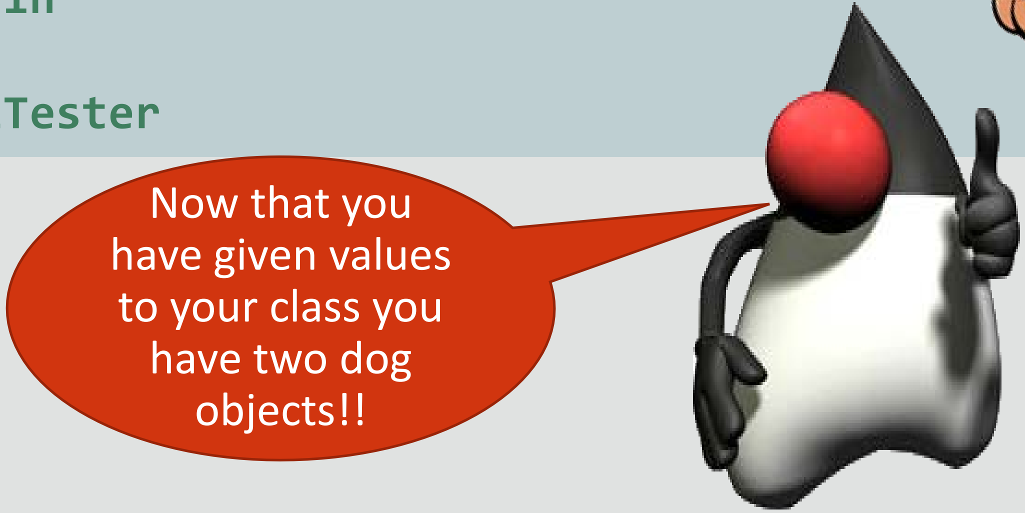


Name	Ace
Breed	Beagle
weight	45.6

Name	Bailey
Breed	Boerboel
Bark noise	arf-arf
weight	80.2

Object/Instance Fields Suggested Solution

```
package animalshop;

public class AnimalTester {

   public static void main(String[] args) {
      Dog dog1 = new Dog("Ace", "Beagle", 45.6);
      Dog dog2 = new Dog("Bailey", "Boerboel", "arf-arf", 80.2);

   }//end method main

}//end class AnimalTester
```


Now that you have given values to your class you have two dog objects!!

Object/Instance Fields Task Continued

- Create method calls for both dog objects so that all of their information is displayed to screen

13. Use the instance field getter methods to produce the following output:
    - Dog name : Bailey
    - Dog breed : Boerboel
    - Bark noise: arf-arf
    - Dog weight: 80.2

Object/Instance Fields Suggested Solution

```
        System.out.println("Dog name   : " + dog1.getName());
        System.out.println("Dog breed  : " + dog1.getBreed());
        System.out.print("Bark noise: ");
        dog1.bark();
        System.out.println("Dog weight: " + dog1.getWeight());

        System.out.println("Dog name   : " + dog2.getName());
        System.out.println("Dog breed  : " + dog2.getBreed());
        System.out.print("Bark noise: ");
        dog2.bark();
        System.out.println("Dog weight: " + dog2.getWeight());
    }//end method main

}//end class AnimalTester
```

Superclass versus Subclass

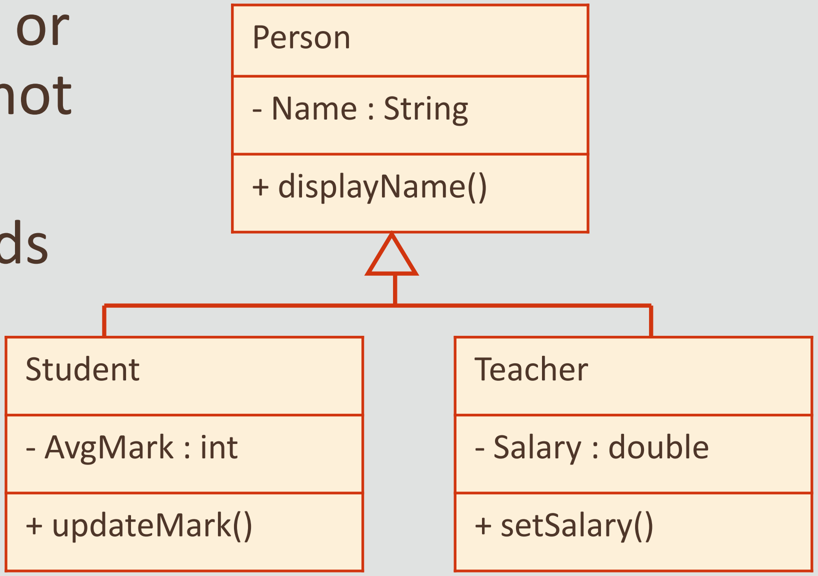
- Classes can derive from or evolve out of parent classes, which means they contain the same methods and fields as their parents, but can be considered a more specialized form of their parent classes
- The difference between a subclass and a superclass is as follows:

Superclass	**Subclass**
The more general class from which other classes derive their methods and data	The more specific class that derives or inherits from another class (the superclass)

ORACLE Academy

Superclass versus Subclass

- Superclasses:
  - Contain methods and fields that are passed down to all of their subclasses
- Subclasses:
  - Inherit methods and fields from their superclasses
  - May define additional methods or fields that the superclass does not have
  - May redefine (override) methods inherited from the superclass



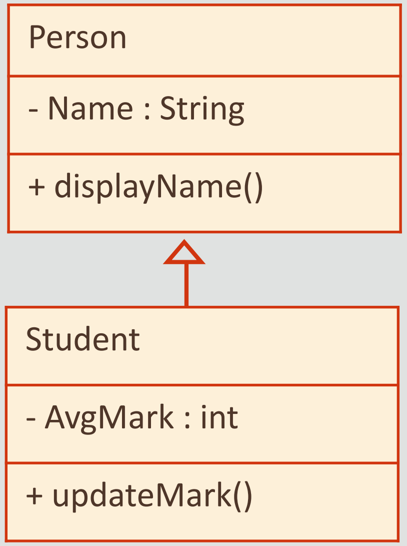
extends Keyword

- In Java, you have the choice of which class you want to inherit from by using the keyword extends
- The keyword extends allows you to designate the superclass that has methods you want to inherit, or whose methods and data you want to extend
- For example, to inherit methods from the Shape class, use extends when the Rectangle class is created

```
public class Rectangle extends Shape
{
   //code
}
```

More About Inheritance

- Inheritance is a one-way street
- Subclasses inherit from superclasses, but superclasses cannot access or inherit methods and data from their subclasses
- This is just like how parents don't inherit genetic traits like hair color or eye color from their children
- A student has access to the name field in the superclass but the Person class has no knowledge of the Student's average mark



Object: The Highest Superclass

- Every superclass implicitly extends the class Object
- Object:
  - Is considered the highest and most general component of any hierarchy. It is the only class that does not have a superclass.
  - Contains very general methods which every class inherits
- The Java API has more information on the object class:

https://docs.oracle.com/javase/10/docs/api/java/lang/Object.html

Java supports only single inheritance. That means that it can only extend one class, and not multiple classes

Encapsulation

- Encapsulation is a fundamental concept in object oriented programming
- Encapsulation means to enclose something into a capsule or container, such as putting a letter in an envelope
- In object-oriented programming, encapsulation encloses, or wraps, the internal workings of a Java instance/object

Private

protected

public

How Encapsulation Works

- Private data fields are hidden from the user of the object
- Public methods can provide access to the private data, but methods hide the implementation
- Encapsulating your data prevents it from being modified by the user or other classes so that the data is not corrupted

```
public class Student{
     private int studentId;
     private String name;
     private String ssn;
     private double gpa;

  public Student(){
  }//end constructor method

  public int getStudentId()
  {
    return studentId;
  }//end method getStudentId

  public void setStudentId(int x)
  {
     studentId = x;
  }//end method setStudentId

}//end class Student
```

ORACLE Academy

Object/Instance Fields Task Continued

14. The animal shop now wants to also stock fish in their shop, the software will have to be updated to reflect this. You will have to create an Animal class (superclass) and a Fish class (subclass, same as Dog)
15. They would like to store the colour of all of their animals. All animals must have their colour stored
16. You cannot store any fish objects in your system unless all of the required values have been provided. Create a fish constructor that reflects this

Object/Instance Fields Task Continued

17. Identify the common fields/methods and place them in the superclass. The fields/methods that are specific to each animal type (dog, fish) should be placed in the appropriate class
18. Display both the fish and dog values to the console
19. A fish can be either a cold or heated water variety
20. Update your driver class to create these objects:

Breed	Goldfish
Water type	cold
colour	red

Name	Bailey
Breed	Boerboel
Bark noise	arf-arf
weight	80.2
colour	brown

Object/Instance Fields Suggested Solution

```
package animalshop;

public class Fish extends Animal{
   private String waterType;

   public Fish(String breed, String waterType, String colour) {
      super(breed, colour);
      this.waterType = waterType;
   }//end constructor method

   public String getWaterType() {
      return waterType;
   }//end method getWaterType

   public void setWaterType(String waterType) {
      this.waterType = waterType;
   }//end method setWaterType

}//end class Fish
```

ORACLE Academy

Object/Instance Fields Suggested Solution

```
package animalshop;
public class Dog extends Animal{
   //instance field declarations
   private String name;
   private String barkNoise = "Woof";
   private double weight;

   public Dog(String name, String breed, double weight, String colour) {
      super(breed, colour);
      this.name = name;
      this.weight = weight;
   }//end constructor method

   public Dog(String name, String breed, String noise, double weight, 
String colour) {
      super(breed, colour);
      this.name = name;
      barkNoise = noise;
      this.weight = weight;
   }//end constructor method
```

Continued on next slide ….

Object/Instance Fields Suggested Solution

```
    public String getName() {
       return name;
    }//end method getName
    public void setName(String name) {
       this.name = name;
    }//end method setName
    public double getWeight() {
       return weight;
    }//end method getWeight
    public void setWeight(double weight) {
       this.weight = weight;
    }//end method setWeight
    public void bark(){
       System.out.println(barkNoise);
    }//end method bark
    public void bark(String barkNoise){
       System.out.println(barkNoise);
    }//end method bark
}//end class Dog
```

ORACLE Academy

Object/Instance Fields Suggested Solution

```
public class Animal {
   private String breed, colour;
   public Animal(String breed, String colour) {
      this.breed = breed;
      this.colour = colour;
   }//end constructor method
   public String getBreed() {
      return breed;
   }//end method getBreed
   public void setBreed(String breed) {
      this.breed = breed;
   }//end method setBreed
   public String getColour() {
      return colour;
   }//end method getColour
   public void setColour(String colour) {
      this.colour = colour;
   }//end method setColour
}//end class Animal
```

Object/Instance Fields Suggested Solution

```
package animalshop;
public class AnimalTester {
   public static void main(String[] args) {
      Dog dog = new Dog("Bailey", "Boerboel", "arf-arf", 80.2, "brown");
      Fish fish = new Fish("Goldfish", "cold", "red");

      System.out.println("Dog name  : " + dog.getName());
      System.out.println("Dog breed : " + dog.getBreed());
      System.out.print("Bark noise: ");
      dog.bark();
      System.out.println("Dog weight: " + dog.getWeight());
      System.out.println("Dog colour: " + dog.getColour());

      System.out.println("Fish breed : " + fish.getBreed());
      System.out.println("Water type : " + fish.getWaterType());
      System.out.println("Fish colour: " + fish.getColour());
   }//end method main
}//end class AnimalTester
```

Now you are ready to take on the Oracle Java Programming course!

Terminology

- Key terms used in this lesson included:
  - Java Primitives
  - Strings
  - Logical and Relational Operators
  - Conditional Statements
  - Program Control
  - Object Classes
  - Constructor and Method Overloading
  - Inheritance
  - Encapsulation

Summary

- In this lesson, you should have learned how to:
  - Use Java Primitives
  - Use of Strings
  - Use Logical and Relational Operators
  - Use Conditional Statements
  - Use Program Control
  - Understand Object Classes
  - Understand Constructor and Method Overloading
  - Understand Inheritance
  - Understand Encapsulation

ORACLE
Academy