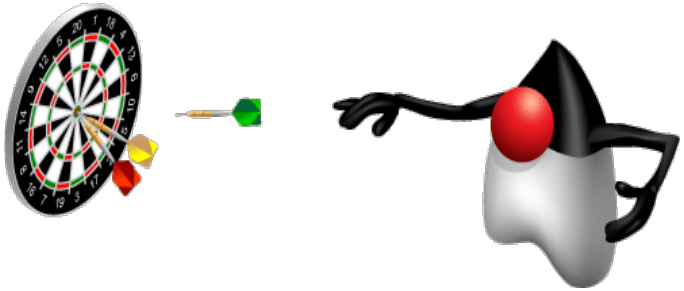
Java Syntax and Class Review

Objectives

After completing this lesson, you should be able to do the following:

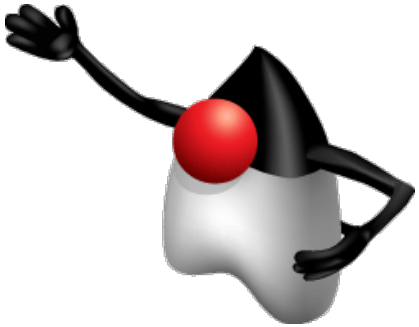
- Create simple Java classes
  - Create primitive variables
  - Use operators
  - Create and manipulate strings
  - Manage Flow Control:
    - Use `if-else` and `switch` statements
    - Iterate with loops: `while`, `do-while`, `for`, enhanced `for`
  - Create arrays
- Use Java fields, constructors, and methods
- Use `package` and `import` statements



Java Language Review

This lesson is a review of fundamental Java and programming concepts. It is assumed that students are familiar with the following concepts:

- The basic structure of a Java class
- Program block and comments
- Variables
- Basic `if-else` and `switch` branching constructs
- Iteration with `for` and `while` loops



Java Class Structure

```
package <package_name>;

import <other_packages>;

public class ClassName {
   <variables(also known as fields)>;

   <constructor(s)>;

   <other methods>;
}
```

A Simple Class

A simple Java class with a `main` method:

```
public class Simple {

    public static void main(String args[]){

    }
}
```

Java Naming Conventions

```
1 public class CreditCard {
2     public final int VISA = 5001;
3     public String accountName;
4     public String cardNumber;
5     public Date expDate;
6
7     public double getCharges(){
8         // ...
9    }
10
11    public void disputeCharge(String chargeId, float amount){
12        // ...
13    }
14}
```

Class names are nouns in upper camel case.

Constants should be declared in all uppercase. letters

Variable names are short but meaningful in lower camel case.

Methods should be verbs, in lower camel case.

How to Compile and Run

Java class files must be compiled before running them.
To compile a Java source file, use the Java compiler (`javac`).

```
javac -cp <path to other classes> -d <compiler output
path> <path to source>.java
```

- You can use the `CLASSPATH` environment variable to the directory above the location of the package hierarchy.
- After compiling the source `.java` file, a `.class` file is generated.
- To run the Java application, run it using the Java interpreter (`java`):

```
java -cp <path to other classes> <package
name>.<classname>
```

How to Compile and Run: Example

- Assume that the class shown in the notes is in the directory `test` in the path `/home/oracle`:

```
$ javac HelloWorld.java
```

- To run the application, you use the interpreter and the class name:

```
$ java HelloWorld
Hello World
```

- The advantage of an IDE like NetBeans is that management of the class path, compilation, and running the Java application are handled through the tool.

Code Blocks

- Every class declaration is enclosed in a code block.
- Method declarations are enclosed in code blocks.
- Java fields and methods have block (or class) scope.
- Code blocks are defined in braces:

```
{   }
```

Example:

```
public class SayHello {
    public static void main(String[] args) {
        System.out.println("Hello world");
    }
}
```

Primitive Data Types

Integer	Floating Point	Character	True False
byte short int long	float double	char	boolean
1, 2, 3, 42 7L 0xff 0b or 0B	3.0 22.0F .3337F 4.022E23	'a' '\u0061' '\n'	true false

Append uppercase or lowercase "`L`" or "`F`" to the number to specify a long or a float number.

Numeric Literals

- Any number of underscore characters (_) can appear between digits in a numeric field.
- This can improve the readability of your code.

```
long creditCardNumber = 1234_5678_9012_3456L;
long socialSecurityNumber = 999_99_9999L;
long hexBytes = 0xFF_EC_DE_5E;
long hexWords = 0xCAFE_BABE;
long maxLong = 0x7fff_ffff_ffff_ffffL;
byte nybbles = 0b0010_0101;
long bytes = 0b11010010_01101001_10010100_10010010;
```

Operators

- Simple assignment operator
  - = Simple assignment operator
- Arithmetic operators
  - + Additive operator (also used for String concatenation)
  - – Subtraction operator
  - * Multiplication operator
  - / Division operator
  - % Remainder operator
- Unary operators
  - + Unary plus operator; indicates positive
  - – Unary minus operator; negates an expression
  - ++ Increment operator; increments a value by 1
  - -- Decrement operator; decrements a value by 1
  - ! Logical complement operator; inverts the value of a boolean

Logical Operators

- Equality and relational operators
  - == Equal to
  - != Not equal to
  - > Greater than
  - >= Greater than or equal to
  - < Less than
  - <= Less than or equal to
- Conditional operators
  - && Conditional-AND
  - || Conditional-OR
  - ?: Ternary (shorthand for `if-then-else` statement)
- Type comparison operator
  - `instanceof` Compares an object to a specified type

`if else` Statement

```
1 public class IfElse {
2
3     public static void main(String args[]){
4         long a = 1;
5         long b = 2;
6
7         if (a == b){
8             System.out.println("True");
9         } else {
10            System.out.println("False");
11        }
12
13     }
14 }
```

`switch` Statement

```
1 public class SwitchStringStatement {
2    public static void main(String args[]){
3
4        String color = "Blue";
5        String shirt = " Shirt";
6
7        switch (color){
8            case "Blue":
9                shirt = "Blue" + shirt;
10                break;
11            case "Red":
12                shirt = "Red" + shirt;
13                break;
14            default:
15                shirt = "White" + shirt;
16        }
17
18        System.out.println("Shirt type: " + shirt);
19    }
20 }
```

`while` Loop

```
package com.example.review;

public class WhileTest {

    public static void main(String args[]) {
        int x = 10;
        while (x < 20) {
            System.out.print("value of x : " + x);
            x++;
            System.out.print("\n");
        }
    }
}
```

expression returning boolean value

do-while Loop

```
package com.example;

public class DoWhileTest {

    public static void main(String args[]) {

        int x = 30;

        do {
            System.out.print("value of x : " + x);
            x++;
            System.out.print("\n");
        } while (x < 20);
    }
}
```

expression returning boolean value

`for` Loop

```
1 public class ForLoop {
2
3     public static void main(String args[]){
4
5         for (int i = 0; i < 9;  i++ ){
6             System.out.println("i: " + i);
7         }
8
9     }
10 }
```

Arrays and `for-each` Loop

```
1 public class ArrayOperations {
2     public static void main(String args[]){
3
4         String[] names = new String[3];
5
6         names[0] = "Blue Shirt";
7         names[1] = "Red Shirt";
8         names[2] = "Black Shirt";
9
10         int[] numbers = {100, 200, 300};
11
12         for (String name:names){
13             System.out.println("Name: " + name);
14         }
15
16         for (int number:numbers){
17             System.out.println("Number: " + number);
18         }
19     }
20 }
```

Arrays are objects. Array objects have a final field length.

Strings

```
1 public class Strings {
2
3     public static void main(String args[]){
4
5         char letter = 'a';
6
7         String string1 = "Hello";
8         String string2 = "World";
9         String string3 = "";
10        String dontDoThis = new String ("Bad Practice");
11
12        string3 = string1 + string2; // Concatenate strings
13
14        System.out.println("Output: " + string3 + " " + letter);
15
16    }
17 }
```

String Operations: `StringBuilder`

```
public class StringOperations {

    public static void main(String arg[]) {

        StringBuilder sb = new StringBuilder("hello");
        System.out.println("string sb: " + sb);
        sb.append(" world");
        System.out.println("string sb: " + sb);

        sb.append("!").append(" are").append(" you?");
        System.out.println("string sb: " + sb);

        sb.insert(12, " How");
        System.out.println("string sb: " + sb);

        // Get length
        System.out.println("Length: " + sb.length());

        // Get SubString
        System.out.println("Sub: " + sb.substring(0, 5));
    }
}
```

A Simple Java Class: `Employee`

A Java class is often used to represent a concept.

```
package com.example.domain;
public class Employee {
    public int empId;
    public String name;
    public String ssn;
    public double salary;

    public Employee () {
    }

     public int getEmpId () {
        return empId;
    }
}
```

class declaration

fields

a constructor

a method

Methods

When a class has data fields, a common practice is to provide methods for storing data (setter methods) and retrieving data (getter methods) from the fields.

```
1 package com.example.domain;
2 public class Employee {
3     public int empId;
4     // other fields...
5     public void setEmpId(int empId) {
6         this.empId = empId;
7     }
8     public int getEmpId() {
9         return empId;
10    }
11    // getter/setter methods for other fields...
12 }
```

Often a pair of methods to set and get the current field value.

Creating an Instance of a Class

To construct or create an instance (object) of the `Employee` class, use the `new` keyword.

```
/* In some other class, or a main method */
Employee emp = new Employee();
emp.empId = 101;    // legal if the field is public,
                    // but not good OO practice
emp.setEmpId(101); // use a method instead
emp.setName("John Smith");
emp.setSsn("011-22-3467");
emp.setSalary(120345.27);
```

Invoking an instance method.

- In this fragment of Java code, you construct an instance of the `Employee` class and assign the reference to the new object to a variable called `emp`.
- Then you assign values to the `Employee` object.

Constructors

```
public class Employee {
    public Employee() {
    }
}
```

A simple no-argument (no-arg) constructor

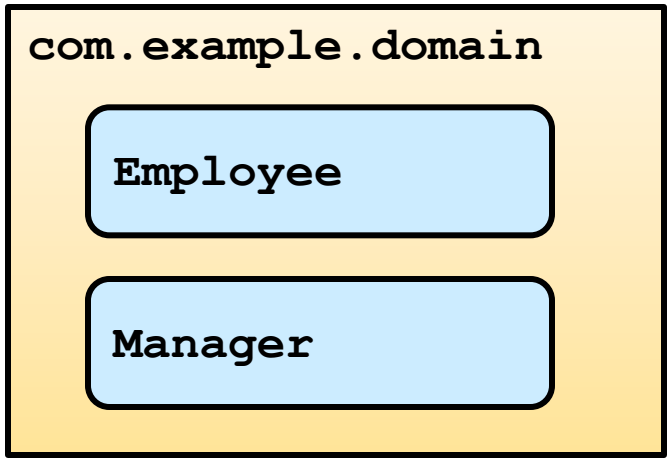
```
Employee emp = new Employee();
```

- A constructor is used to create an instance of a class.
- Constructors can take parameters.
- A constructor is declared with the same name as its class.

`package` Statement

- The `package` keyword is used in Java to group classes together.
- A package is implemented as a folder and, like a folder, provides a *namespace* to a class.

namespace view



folder view

```
+com
|_+example
  |_+domain
    |_+Employee.java
    |_+Manager.java
```

Always declare a package!

`import` Statements

The `import` keyword is used to identify classes you want to reference in your class.

- The `import` statement provides a convenient way to identify classes that you want to reference in your class.

```
import java.util.Date;
```

- You can import a single class or an entire package:

```
import java.util.*;
```

- You can include multiple `import` statements:

```
import java.util.Date;
import java.util.Calendar;
```

- It is good practice to use the full package and class name rather than the wildcard `*` to avoid class name conflicts.

`import` Statements

- `import` statements follow the package declaration and precede the class declaration.
- An `import` statement is not required.
- By default, your class always imports `java.lang.*`
- You do not need to import classes that are in the same package:

```
package com.example.domain;
import com.example.domain.Manager; // unused import
```

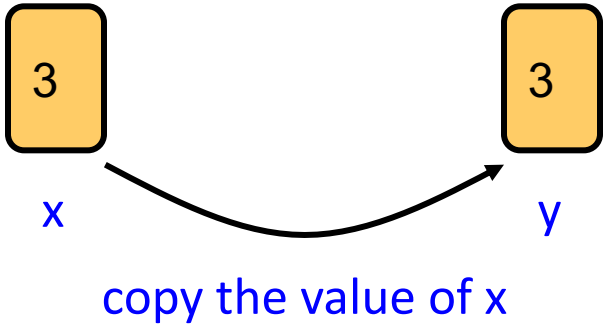
Java Is Pass-By-Value

The Java language (unlike C++) uses pass-by-value for all assignment operations.

- To visualize this with primitives, consider the following:

```
int x = 3;
int y = x;
```

- The value of `x` is copied and passed to `y`:



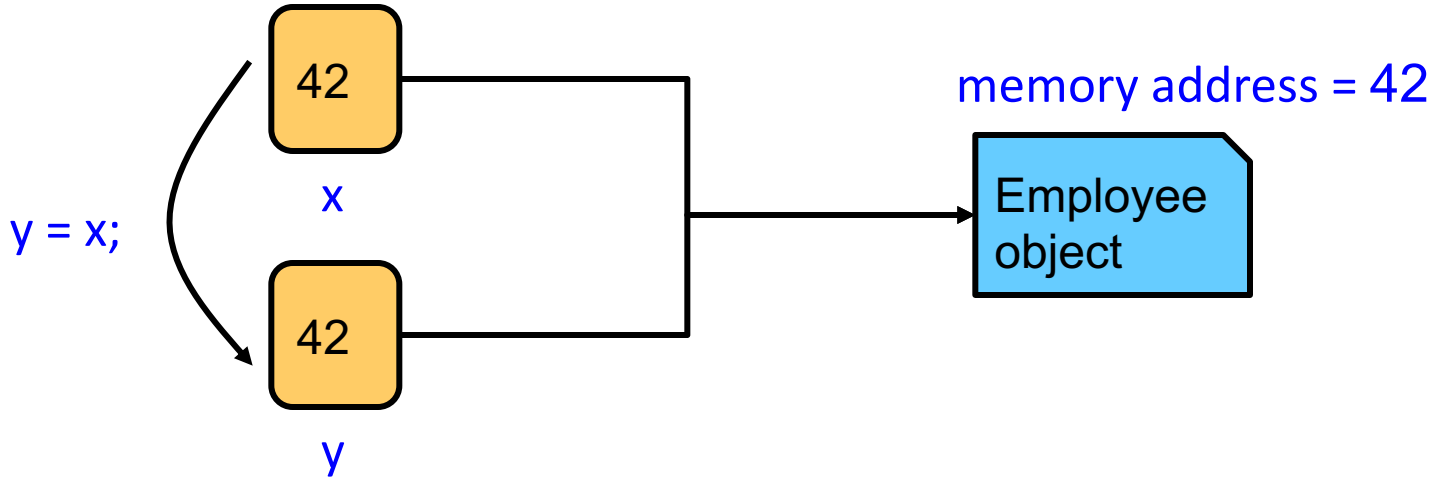
- If `x` is later modified (for example, `x = 5;`), the value of `y` remains unchanged.

Pass-By-Value for Object References

For Java objects, the *value* of the right side of an assignment is a reference to memory that stores a Java object.

```
Employee x = new Employee();
Employee y = x;
```

- The reference is some address in memory.



- After the assignment, the value of `y` is the same as the value of `x`: a reference to the same `Employee` object.

Objects Passed as Parameters

```
 4 public class ObjectPassTest {
 5     public static void main(String[] args) {
 6         ObjectPassTest test = new ObjectPassTest();
 7         Employee x = new Employee ();
 8         x.setSalary(120_000.00);
 9         test.foo(x);
10         System.out.println ("Employee salary: "
11           + x.getSalary());
12     }
13
14     public void foo(Employee e){
15       e.setSalary(130_000.00);
16       e = new Employee();
17       e.setSalary(140_000.00);
18     }
```

salary set to 120_000

What will `x.getSalary()` return at the end of the `main()` method?

Garbage Collection

When an object is instantiated by using the `new` keyword, memory is allocated for the object. The scope of an object reference depends on where the object is instantiated:

```
public void someMethod() {
    Employee e = new Employee();
    // operations on e
}
```

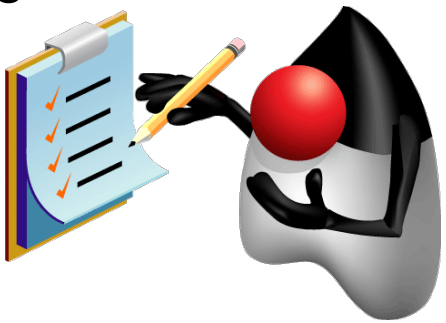
Object `e` scope ends here.

- When `someMethod` completes, the memory referenced by `e` is no longer accessible.
- Java's garbage collector recognizes when an instance is no longer accessible and eligible for collection.

Summary

In this lesson, you should have learned how to:

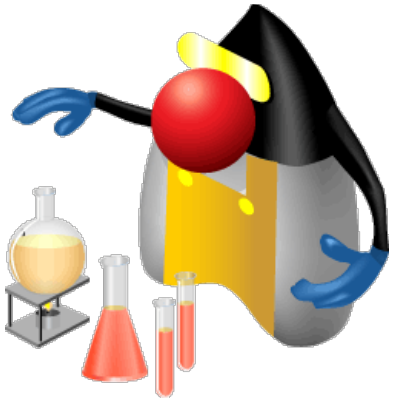
- Create simple Java classes
  - Create primitive variables
  - Use Operators
  - Manipulate Strings
  - Use `if-else` and `switch` branching statements
  - Iterate with loops
  - Create arrays
- Use Java fields, constructors, and methods
- Use `package` and `import` statements



Practice 2-1 Overview: Creating Java Classes

This practice covers the following topics:

- Creating a Java class using the NetBeans IDE
- Creating a Java class with a `main` method
- Writing code in the body of the `main` method to create an instance of the `Employee` object and print values from the class to the console
- Compiling and testing the application by using the NetBeans IDE



Quiz

Which is the printed result in the following fragment?

```
public float average (int[] values) {
    float result = 0;
    for (int i = 1; i < values.length; i++)
        result += values[i];
    return (result/values.length);
}
// ... in another method in the same class
int[] nums = {100, 200, 300};
System.out.println (average(nums));
```

a. 100.00
b. 150.00
c. 166.66667
d. 200.00

Quiz

In the following fragment, which two statements are false?

```
package com.oracle.test;
public class BrokenClass {
    public boolean valid = "false";
    public String s = "A new string";
    public int i = 40_000.00;
    public BrokenClass() { }
}
```

a. An `import` statement is missing.

b. The `boolean valid` is assigned a String.

c. `String s` is created.

d. `BrokenClass` method is missing a `return` statement.

e. You need to create a new `BrokenClass` object.

f. The integer value `i` is assigned a double.

Quiz

What is displayed when the following code snippet is compiled and executed?

```
String s1 = new String("Test");
String s2 = new String("Test");
if (s1==s2)
    System.out.println("Same");
if (s1.equals(s2))
    System.out.println("Equals");
```

a. Same
b. Equals
c. Same Equals
d. Compiler Error