Altar Code Labs

Java Basics

JSE 1/4

UP ACADEMY
LEARNING FOR SUCCESS

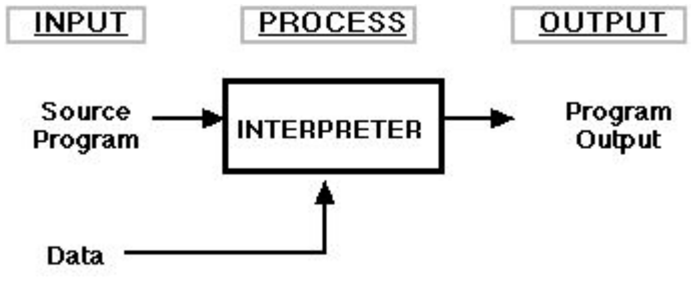
CONTENTS

Java Basics

- *Compilation vs Interpretation*
- *Java basics*
- *Eclipse IDE*

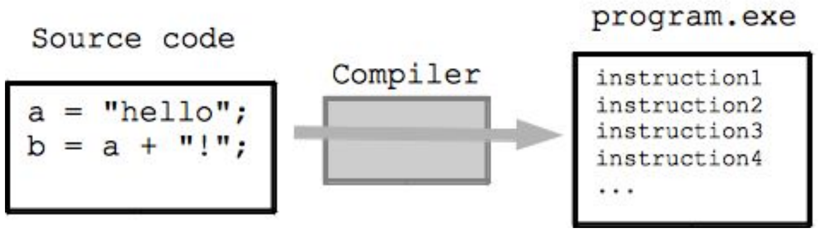
Interpretation

- An **interpreted language** is a programming language for which most of its implementations **execute instructions directly**, without previously compiling a program into machine-language instructions
- The interpreter **executes the program directly**, translating each statement into a sequence of **one or more subroutines** already compiled into machine code



Compilation

- **Translation** of source code into object code by a compiler
- A compiler is a **computer program** (or a set of programs) that **transforms** source code written in a programming language (the source language) into **another computer language** (the target language), with the latter often having a binary form known as object code



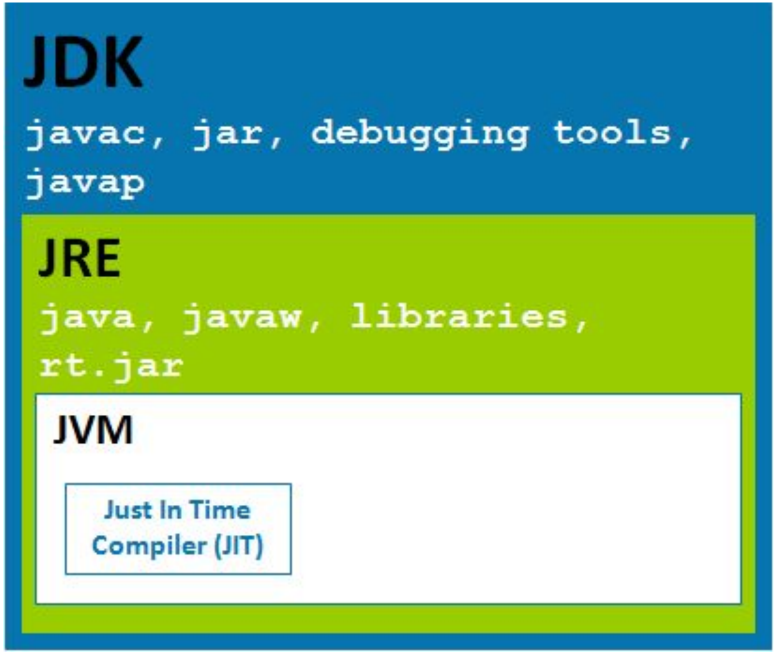
Once upon a time...

- A team of highly skilled software developers at **Sun** (under the leadership of **James Gosling**) wanted to create a programming language that enabled consumer devices with **different CPUs** (Central Processing Units) to share the same software enhancements
- The World Wide Web was becoming popular and the Sun team recognized that the **Oak** language was perfect for developing web multimedia components to enhance webpages. These small applications, called **applets**, became the **initial use of the Oak language**, and programmers using the Internet adopted what became the **Java programming language**
- The turning point for Java came in **1995**, when **Netscape incorporated Java into its browser**

Java artifacts

- Class
- Object
- .java file
- .class file
- Packages
- JVM
- JRE
- Byte code
- JAR
- Methods (Main method)
- Garbage collector
- JIT compilation
- Javac
- JDK

Java


JDK
javac, jar, debugging tools, javap
JRE
java, javaw, libraries, rt.jar
JVM
Just In Time Compiler (JIT)

Java hello world

```
public class HelloWorld {
        public static void main(String[] args) {
                System.out.println("Hello! What’s your name?");
        }
}
```

Classes and Objects

- A class is the way you define an object.
- Classes are descriptive categories, templates or blueprints.
  - Car could be a class defining all cars to have a car ID, model, year, description, and a price.
- Objects are unique instances of classes:
  - A car with ID 1, model VW Beetle, year 1970, price 10000 USD
  - Another car with ID 2, model BMW 2002, year 1980, price 9000 USD

Class components

- Classname
- Fields
- Constructors
- Methods

- Examples:
  - Car
  - carId
  - color
  - Car()
  - changeColor()
  - getCarId()

Class structure

```
package org.altar.session1;

import org.altar.session1.types.Color;

public class Car {
   private int carId = 0;
   private Color color;

   public int getCarId() {
      return carId;
   }
   public void setCarId(int carId) {
      this.carId = carId;
   }
   public void setColor(Color color) {
      this.color = color;
   }
}
```

Comments

- **Single line:**
```
public int carID = 0;  // Default ID
```

- **Multiple Line:**
```
/*
The following lines of code will clean all the duplicate items in the list
and will populate it with the replacing elements
*/
```

Method syntax

```
[modifiers] return_type method_identifier ([arguments]){
    method_code
}
```

Ex:

```
public int getCarId() {
    return carId;
}
```

Access modifiers : public, protected, default (empty), private
Other modifiers : static, final, synchronized, ...
return_type : void, int, double, String, Car, ...

Test class with a main method

```
package org.altar.session1.test;

import org.altar.session1.Car;

public class CarTest {
        public static void main(String[] args) {
                Car myFirstCar = new Car();
                myFirstCar.setCarId(1);
                System.out.println("My car "+myFirstCar.getCarId() );
        }
}
```

Test class that reads from keyboard

```
package org.altar.session1.test;

import java.util.Scanner;

public class ScanTest {
   public static void main(String[] args) {
      System.out.println("Hello! What's your name?");

      Scanner scanner = new Scanner(System.in);
      String name = scanner.nextLine();
      scanner.close(); // This line is not mandatory, but it is a good practice

      System.out.println("Hello " + name + "!!!");
   }
}
```

Platform independent programs

- javac Person.java *(name of the file)*
- java Person *(name of the class)*
- You use javac to compile a text file
- Everyone can use a free JRE to execute your compiled class file, on Windows, Linux, Mac Os X, Solaris, etc.

Setup and use

- Download Java Development Kit
- Set your OS Path to the installed dir
- Compile and Run your programs
- Integrated Development Environment (IDE) is a tool that can assist you with your Java application development.
  - Eclipse, Intellij, Netbeans, JDeveloper
  - Useful for automating task, code completion, smart editor, source code control, error detection, debugging…

Eclipse Tutorial

Java Variables

- *Variables*
- *Primitive types*
- *Objects*
- *Arrays*

Variables

- A variable holds data.
- Can be a mathematical expression or a primitive data type or a reference to an object.
- Syntax for fields and local variables : [modifiers] type identifier [= value];
- Example:

```
        public int carId = 0;
```

Naming

- Must start either with:
    - an uppercase or lowercase letter
    - an underscore “_”
    - a dollar sign “$”

- Cannot contain punctuation, spaces or dashes

- Java keywords cannot be used

- Good practice:
    - Use camelCase
    - Choose clear names that indicate the intent of the variable.

Declaring and Initializing Variables in One Line of Code

- Syntax:

  type identifier = value [, identifier = value];

- Example:

  double discount = 0.0, myPrice = 0.0;

Declaring and Initializing Variables

- Assign literal values
  boolean isInDebt = false;
  int carID = 0;

- Assign the value of one other variable
  int mySecondCar = carID;

- Assigning the result of an expression to floating point or boolean variables:
  float myPrice = 29.99F;
  boolean isOverPriced = (myPrice > 800.00);

- Assigning the return value of a method call to a variable
  int carId = generateCarID();

Constants

- Is a “variable” that can have a value assigned only once
- *final* keyword
- Example:

  ```
  final int FINANTIAL_DAYS_YEAR = 360;
  ```

- Best practice:
  - Capitalized with words separated by an underscore “_”

Operators

- Standard: + - * /
- Remainder: %
- Increment and Decrement : ++ --
- Pre-decrement (--variable)
  - int i = 2;
  - int j = --i;
  - i is 1, j is 1
- Post-decrement (variable--)
  - int i = 2;
  - int j = i--;
  - i is 1, j is 2
- Minus: - (Example: -5 or -(3+4))

Type Casting

- Assigning a variable or an expression to another variable can lead to a mismatch between the data types of the calculation and the storage location that you are using to save the result
- Issue:
  - int num1 = 53; // 32 bits of memory to hold the value
  - int num2 = 47; // 32 bits of memory to hold the value
  - byte num3; // 8 bits of memory reserved
  - num3 = (num1 + num2); // causes compiler error
- To fix this problem, you can either type cast the right-side data type down to match the left-side data type, or declare the variable on the left side (num3) to be a larger data type, such as an int.
  - byte num3 = (byte) (num1+num2);
  - int num3 = (num1 + num2);
- When a mathematical operation is performed, the result is stored in a temporary internal variable of type int or, if some of the operands is larger, then the variable will have that type. The value is then assigned to the variable created by the developer. Knowing this, it is easy to understand that, in order to the cast to be effective, it must be applied on the final result of the operation.

Primitive Data Types

- Integer types
  - **byte** 8 bits
  - **short** 16 bits
  - **int** 32 bits
  - **long** 64 bits

- Floating point types
  - **float** 32 bits
  - **double** 64 bits

- Textual type
  - **char** 16 bits

- Logical type
  - **boolean** undefined

Floating Point Data Types

- Example of potential problem:
```
float float1 = 37.0;        //compiler error
```
- Example of potential solutions:
  - The F notifies the compiler that 37.0 is a float value:
    ```
    float float1 = 37.0F;
    ```
  - 37.0 is cast to a float type:
    ```
    float float1 = (float) 37.0;
    ```

Objects

- Objects are accessed via references
- Objects are instantiated versions of their class

  Object reference = new Classname();

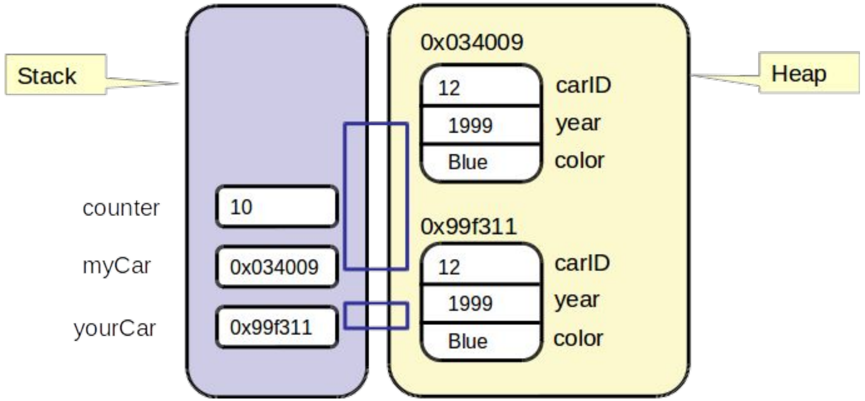
```
Car myCar = new Car();
int myCarId = myCar.carId;
int myCarIdAlternative = myCar.getCarId();
```

```
public class Car {
        public int carId = 0;

        public int getCarId() {
                return carId;
        }
}
```

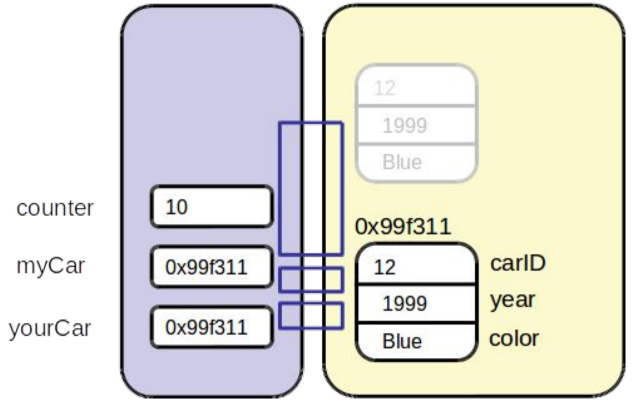
References and Objects In Memory

```
int counter = 10;
Car myCar = new Car();
Car yourCar = new Car();
```



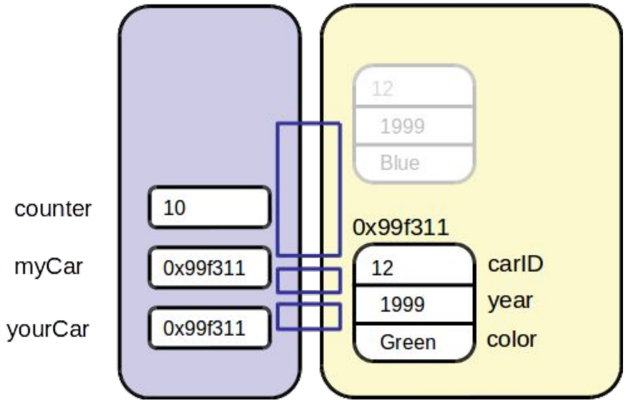
References and Objects In Memory

myCar = yourCar;



References and Objects In Memory

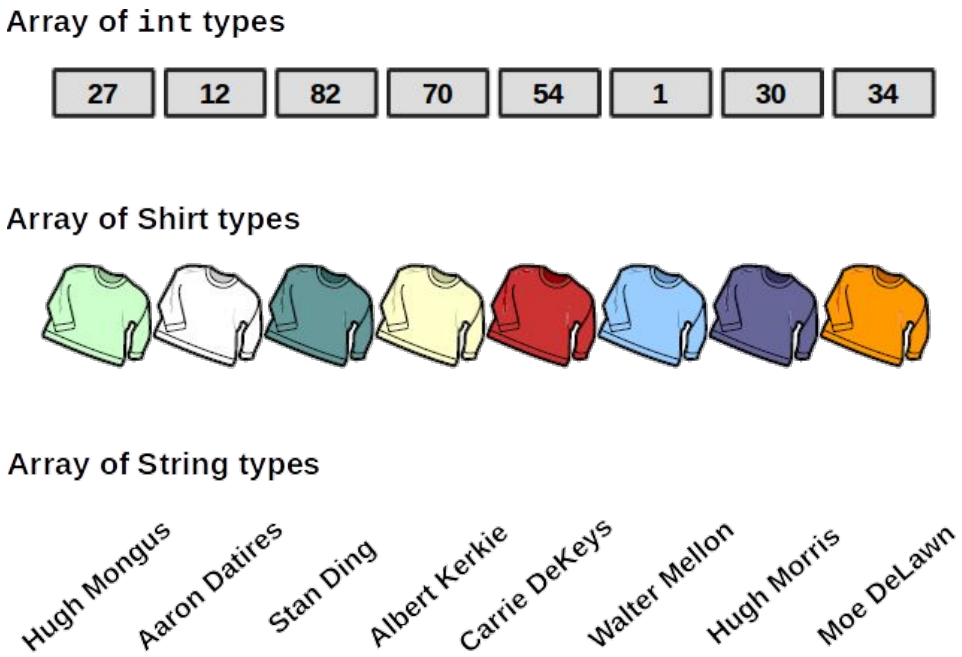
```
myCar.color = "Red";
yourCar.color = "Green";
```



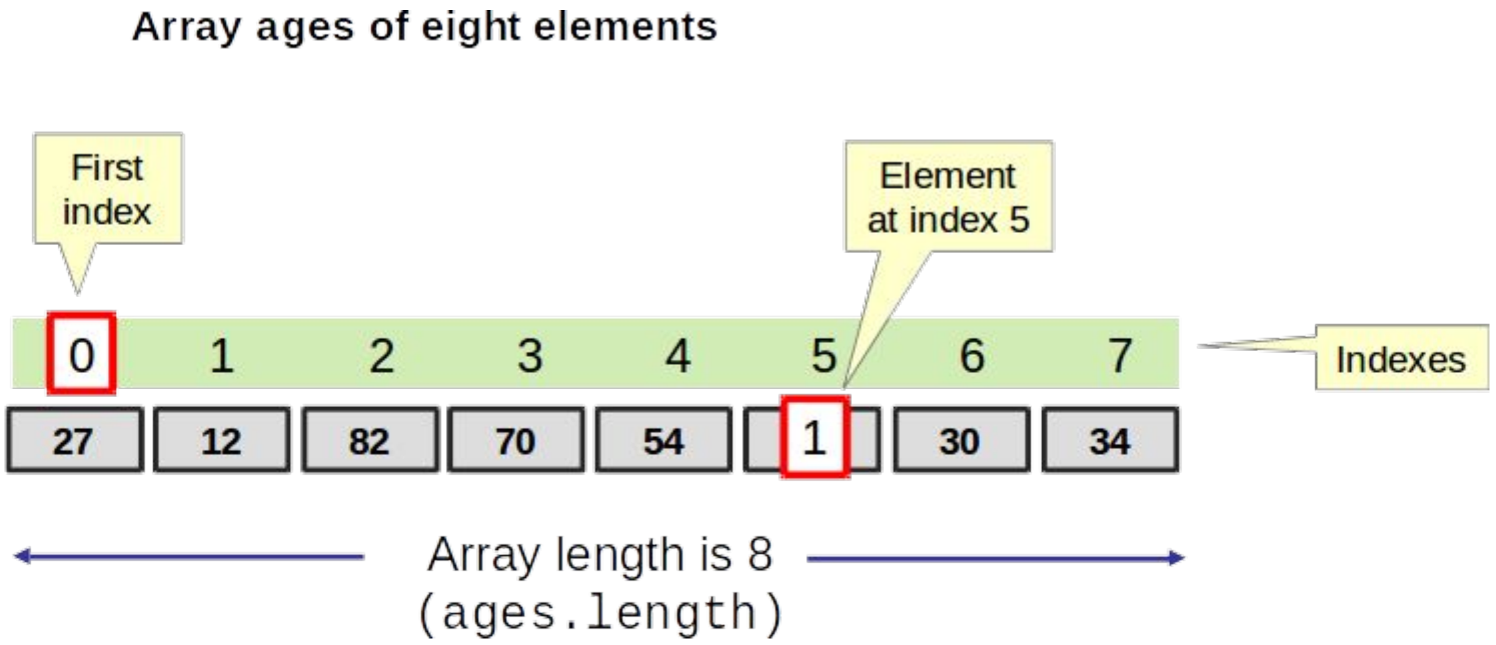
Introduction to Arrays

- An array is a container object that holds a group of values of a single type
- A value in the array can be a primitive or an object type
- The length of an array is established when the array is created
- After creation, the length of an array cannot be changed
- Each item in an array is called an element
- Each element is accessed by a numerical index
- The index of the first element is 0 (zero)

One-Dimensional Arrays



One-Dimensional Arrays



Declaring a One-Dimensional Array

- Syntax:
  - *type [] array_identifier;*
  - *type array_identifier[];*
- Declare arrays of types char and int:
  - char[] status;
  - int ages[];
- Declare arrays of object references of types Shirt and String:
  - Shirt shirts [];
  - String [] names;

Instantiating a One-Dimensional Array

- Syntax:
  - *array_identifier = new type [length];*
- Examples:
  - status = new char [20];
  - ages = new int [5];
  - names = new String [7];
  - shirts = new Shirt [3];

Initializing a One-Dimensional Array

- Syntax:
  - *array_identifier[index] = value;*
- Examples:
  - ages[0] = 19;
  - ages[1] = 42;
  - names[1] = “Ellen”;
  - shirts[3] = new Shirt();

Declaring, Instantiating and Initializing

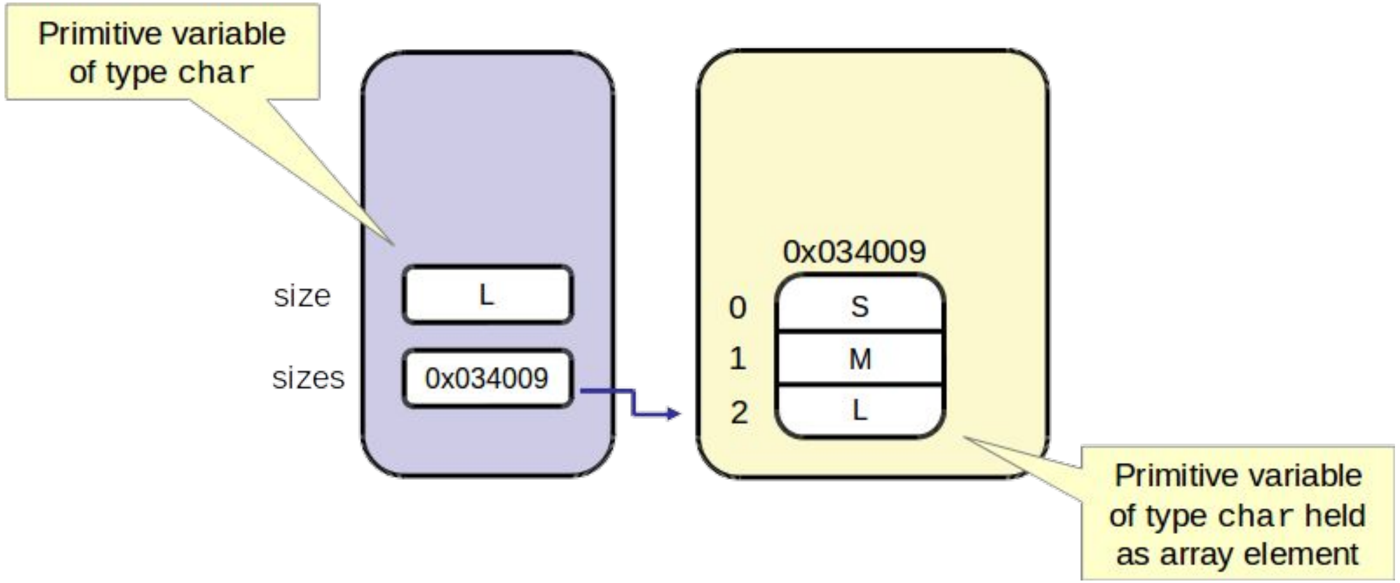
- Syntax:
  - *type [] array_identifier = {comma-separated list of values or expressions};*
- Examples:
  - int [] ages = {19, 42, 92, 33, 46};
  - Shirt [] shirts = {new Shirt(), new Shirt(), new Shirt()};
- Not permitted (IDE will show an error):
  - int [] ages;
  - ages = {19, 42, 92, 33, 46};

Accessing a Value Within an Array

- Setting a value:
  - status[0] = '3';
  - names[1] = "Fred Smith";
  - ages[1] = 19;
  - prices[2] = 9.99F;
- Getting a value:
  - char s = status[0];
  - String name = names [1];
  - int age = ages[1];
  - double price = prices[2];

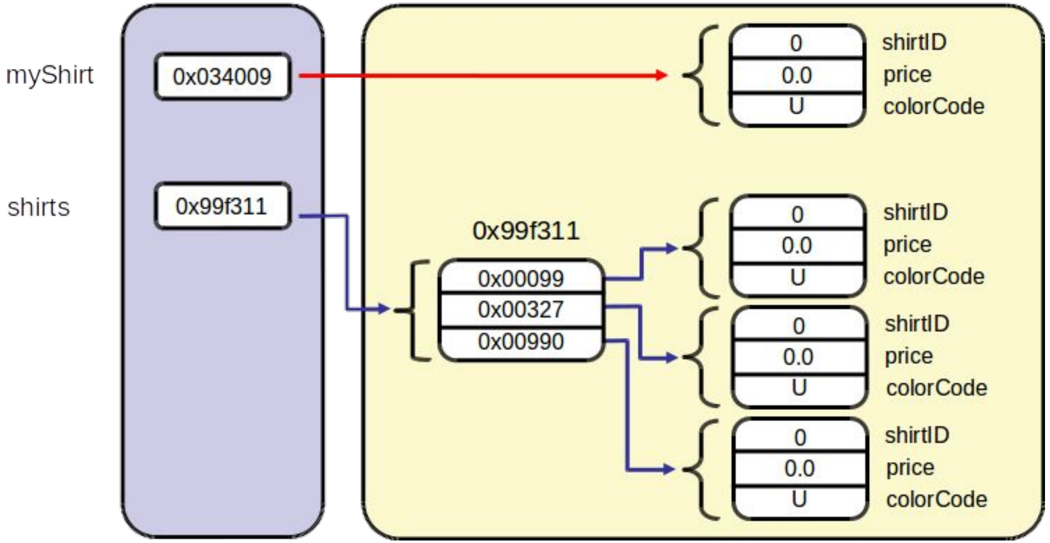
Storing Arrays in Memory

- char size = 'L'
- char[] sizes = {'S', 'M', 'L' };


Primitive variable of type char
size
L
sizes
0x034009
0x034009
0
S
1
M
2
L
Primitive variable of type char held as array element

Storing Arrays of References in Memory

- Shirt myShirt = new Shirt();
- Shirt[] shirts = { new Shirt(), new Shirt(), new Shirt() };



Using the args Array in the main Method

- Parameters can be typed on the command line:
  - > java ArgsTest Hello World!
  - args[0] is Hello
  - args[1] is World!

The second parameter goes into args[1] and so on

The first parameter goes into args[0]

- Code for retrieving the parameters:

```
public class ArgsTest {
    public static void main (String[] args) {
        System.out.println("args[0] is " + args[0]);
        System.out.println("args[1] is " + args[1]);
    }
}
```

Converting String Arguments to Other Types

- Numbers can be typed as parameters:
    - java ArgsTest 2 3
    - Total is: 23
    - Total is: 5

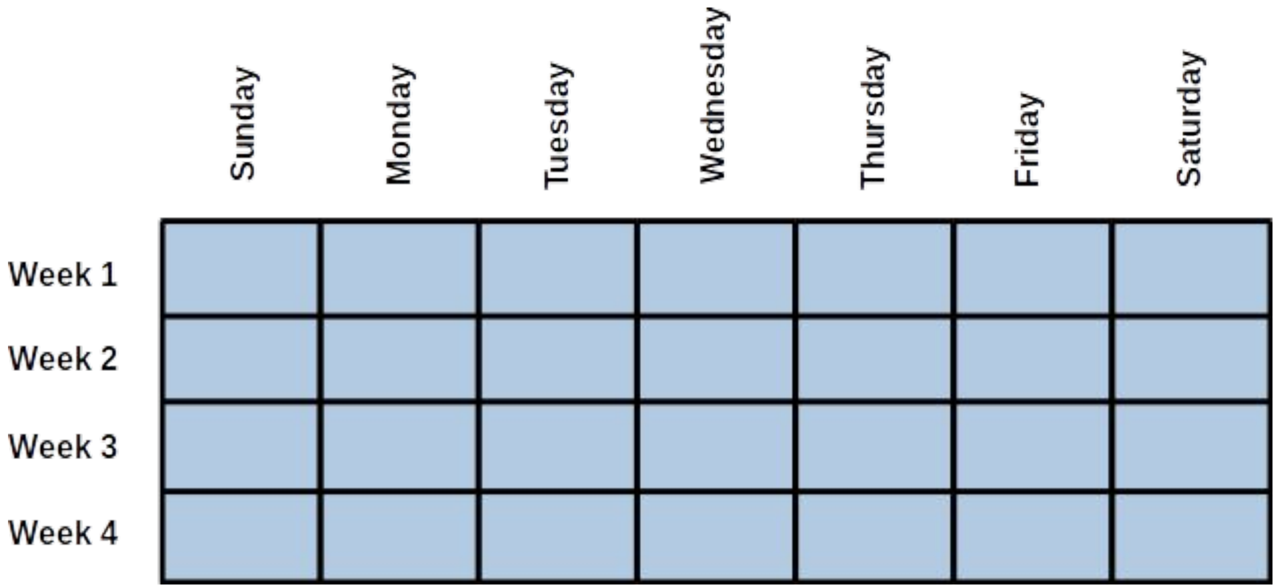
Concatenation, not sum!

- Conversion of String to int:

```
public class ArgsTest {
  public static void main (String[] args) {
     System.out.println("Total is: " + (args[0] + args[1]));

     int arg1 = Integer.parseInt(args[0]);
     int arg2 = Integer.parseInt(args[1]);
     System.out.println("Total is: " + (arg1 + arg2));
  }
}
```

Describing Two-Dimensional Arrays



Declaring a Two-Dimensional Array

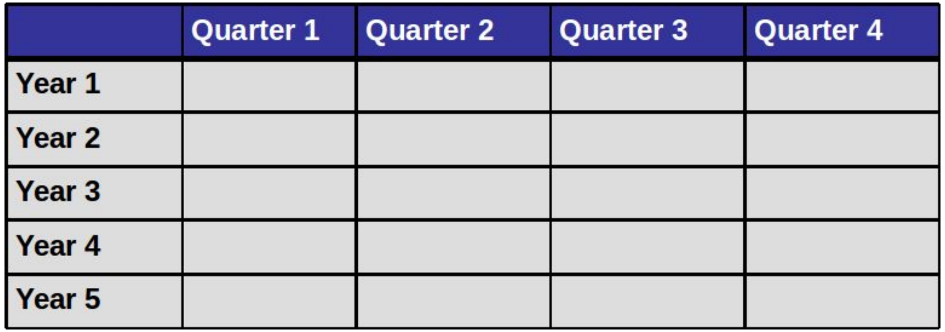
- Syntax:
  - *type [][] array_identifier;*
  - *type [] array_identifier [];*
  - *type array_identifier[][];*

- Example:
  - int [][] yearlySales;
  - int[] yearlySales[];
  - int yearlySales[][];

Instantiating a Two-Dimensional Array

- Syntax:
  - *array_identifier = new type [number_of_arrays] [length];*

- Example:
  - // Instantiates a 2D array: 5 arrays of 4 elements each
  - yearlySales = new int[5][4];



	Quarter 1	Quarter 2	Quarter 3	Quarter 4
Year 1				
Year 2				
Year 3				
Year 4				
Year 5				

Initializing a Two-Dimensional Array

- Example:
  - yearlySales[0][0] = 1000;
  - yearlySales[0][1] = 1500;
  - yearlySales[0][2] = 1800;
  - yearlySales[1][0] = 1000;
  - yearlySales[3][3] = 2000;

	Quarter 1	Quarter 2	Quarter 3	Quarter 4
Year 1	1000	1500	1800	
Year 2	1000			
Year 3				
Year 4				2000
Year 5				

Java Constructs

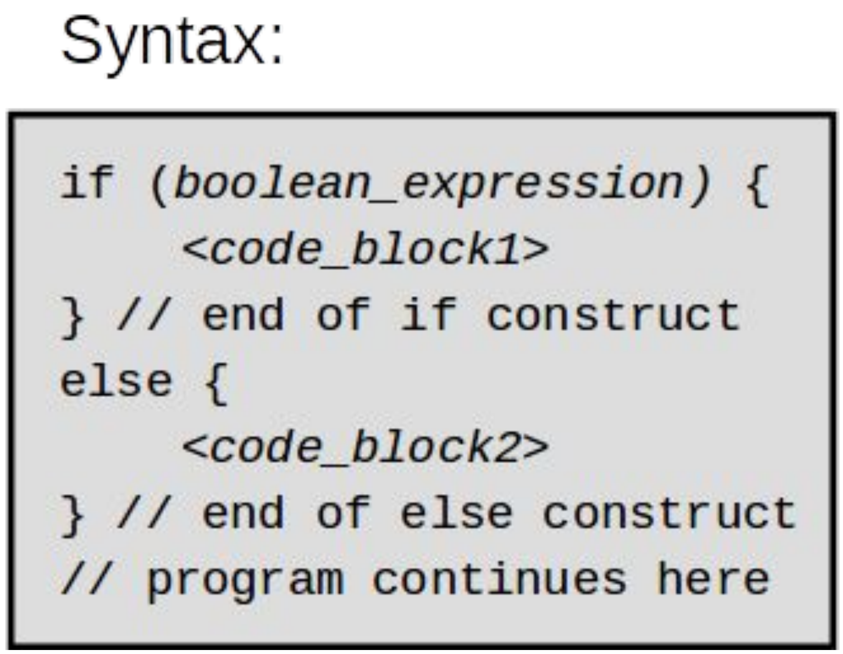
- *Conditions*
- *Loops*
- *String and StringBuilder*
- *Java API doc*

Relational and Boolean Operators

Condition	Operator	Example
Is equal to	==	`int i=1;` `(i == 1)`
Is not equal to	!=	`int i=2;` `(i != 1)`
Is less than	<	`int i=0;` `(i < 1)`
Is less than or equal to	<=	`int i=1;` `(i <= 1)`
Is greater than	>	`int i=2;` `(i > 1)`
Is greater than or equal to	>=	`int i=1;` `(i >= 1)`

Operation	Operator	Example
If one condition AND another condition	&&	`int i = 2;` `int j = 8;` `((i < 1) && (j > 6))`
If either one condition OR another condition	||	`int i = 2;` `int j = 8;` `((i < 1) || (j > 9))`
NOT	!	`int i = 2;` `(!(i < 3))`

If/Else construct



Syntax:

```
if (boolean_expression) {
      <code_block1>
} // end of if construct
else {
      <code_block2>
} // end of else construct
// program continues here
```

If/Else construct

Example:

```
public void calculateNumDays(int month) {
    if (month == 1 || month == 3 || month == 5 || month == 7 ||
         month == 8 || month == 10 || month == 12) {
        System.out.println("There are 31 days in that month.");
    }
    else if (month == 2) {
        System.out.println("There are 28 days in that month.");
    }
    else if (month == 4 || month == 6 || month == 9 || month == 11) {
        System.out.println("There are 30 days in that month.");
    }
    else {
        System.out.println("Invalid month.");
    }
}
```

Ternary Conditional Operator

Operation	Operator	Example
If `someCondition` is true, assign the value of `value1` to result. Otherwise, assign the value of `value2` to result.	`?:`	`someCondition ? value1 : value2`

Example:

value = (value >= 50) ? value - 50 : value;

Switch Construct

Syntax:

```
switch (variable) {
    case literal_value:
         <code_block>
         [break;]
    case another_literal_value:
        <code_block>
         [break;]
    [default:]
         <code_block>
}
```

- Tests against int, short, byte, char, enums and Strings
- Tests against a fixed value known at compile time

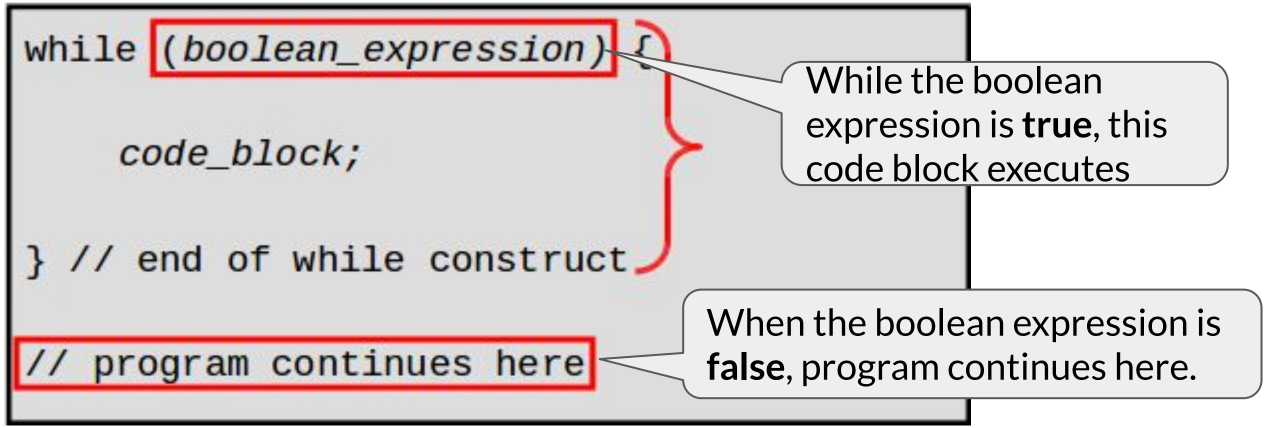
Switch Construct

Example:

```
public void calculateNumDays(int month) {
    switch(month) {
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:
            System.out.println("31 days.");
            break;
        case 4:
        case 6:
        case 9:
        case 11:
            System.out.println("30 days.");
            break;
        default:
            System.out.println("28 days.");
    }
}
```

While Construct

Syntax:



```
while (boolean_expression) {

     code_block;

} // end of while construct

// program continues here
```

While the boolean expression is **true**, this code block executes
When the boolean expression is **false**, program continues here.

While Construct

Example:

```
float square = 4;     // number to find sq root of
float squareRoot = square;    // first guess
while (squareRoot * squareRoot - square > 0.001) {
  squareRoot = (squareRoot + square/squareRoot)/2;
  System.out.println("Next try will be " + squareRoot);
}
System.out.println("Square root of " + square +
       " is " + squareRoot);
```

Result:

```
   Next try will be 2.5
   Next try will be 2.05
   Next try will be 2.0006099
   Next try will be 2.0
   The square root of 4.0 is 2.0
```

Do/While Construct

Syntax:

```
do {
    code_block;
}
while (boolean_expression); // Semicolon is mandatory
```

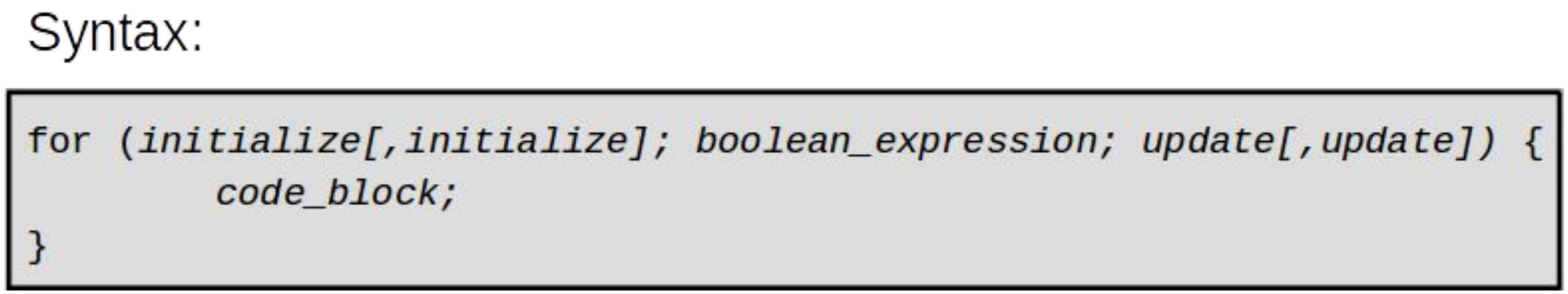
Do/While Construct

Example:

```
setFloor() {
  int desiredFloor = 5;

  do {
    if (currentFloor < desiredFloor) {
      goUp();
    }
    else if (currentFloor > desiredFloor) {
      goDown();
    }
  }
  while (currentFloor != desiredFloor);
}
```

For Construct

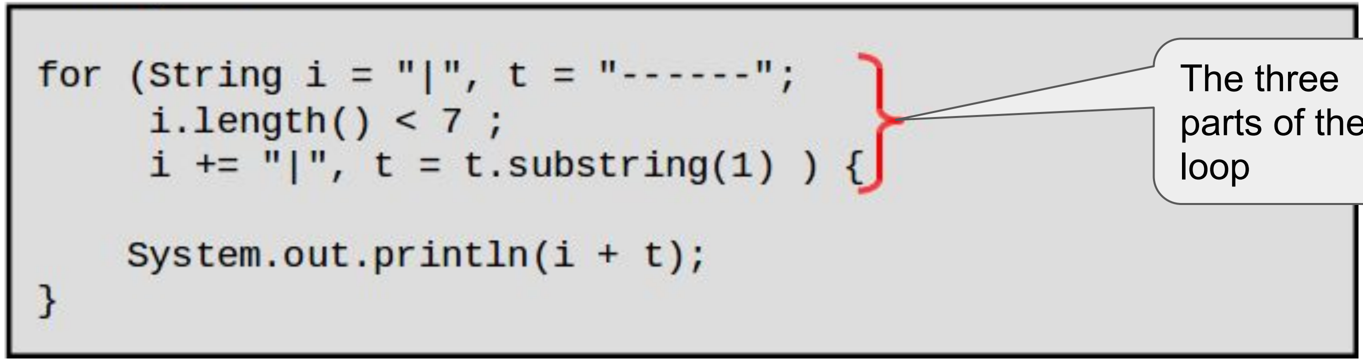


Syntax:

```
for (initialize[,initialize]; boolean_expression; update[,update]) {
        code_block;
}
```

For Construct

Example:

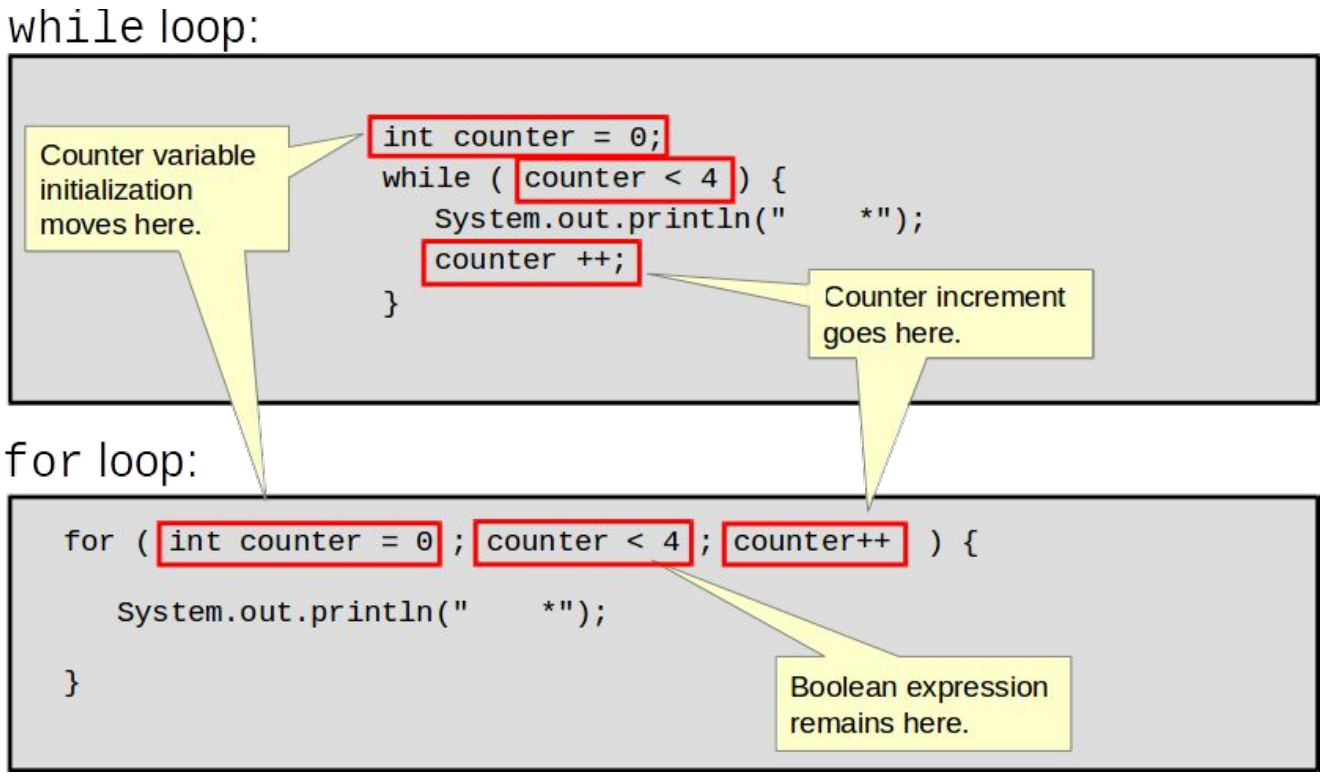


```
for (String i = "|", t = "------";
     i.length() < 7 ;
     i += "|", t = t.substring(1) ) {

    System.out.println(i + t);
}
```

The three parts of the for loop

Comparing While and For Constructs



while loop:

```
int counter = 0;
while ( counter < 4 ) {
   System.out.println("    *");
   counter ++;
}
```

for loop:

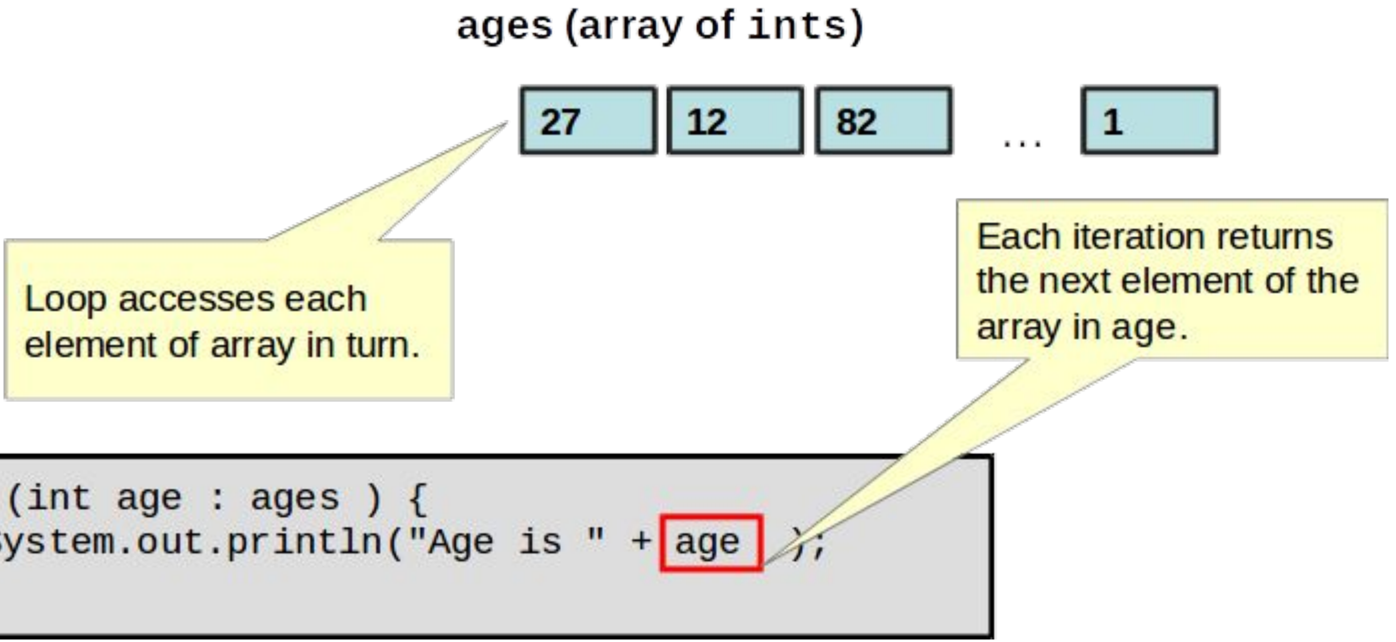
```
for ( int counter = 0 ; counter < 4 ; counter++  ) {

   System.out.println("    *");

}
```

Counter variable initialization moves here.
Counter increment goes here.
Boolean expression remains here.

Enhanced for (foreach) Loop


ages (array of ints)
27
12
82
. . .
1
Loop accesses each element of array in turn.
Each iteration returns the next element of the array in age.

```
for (int age : ages ) {
    System.out.println("Age is " + age );
}
```

You can use either arrays or any object that implements interface Iterable on the enhanced for (foreach) loop

Break with loops

break example:

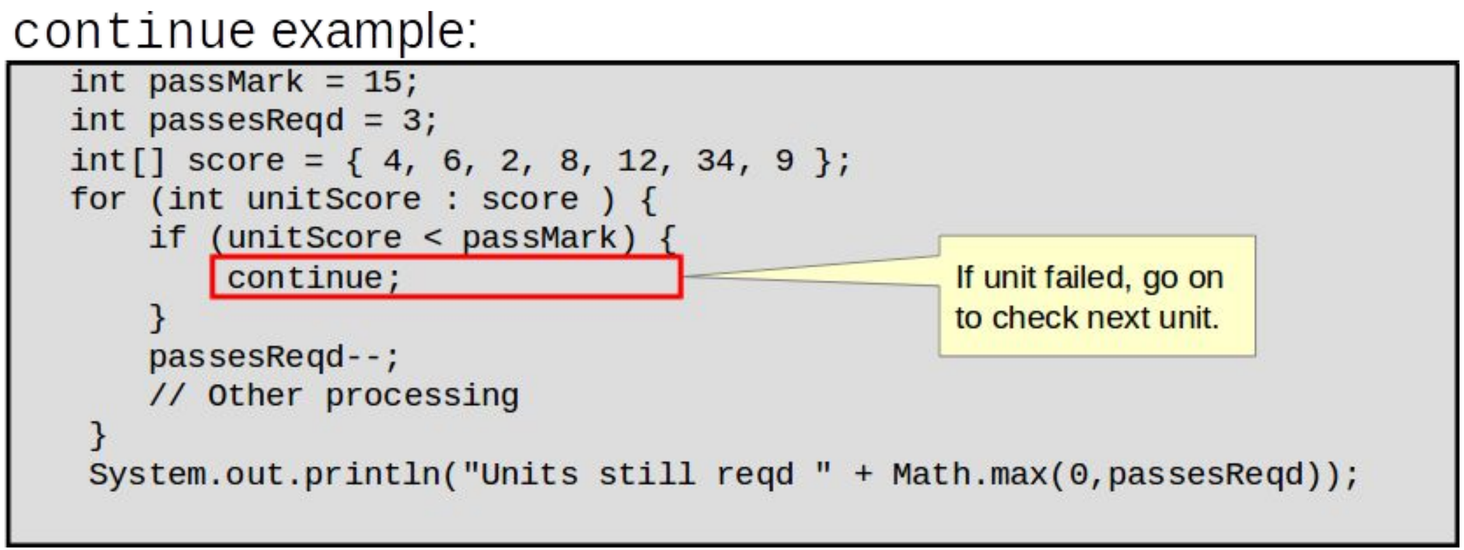
```
int passmark = 12;
boolean passed = false;
int[] score = { 4, 6, 2, 8, 12, 34, 9 };
for (int unitScore : score ) {
    if ( unitScore > passmark ) {
        passed = true;
        break;
    }
}
System.out.println("One or more units passed? " + passed);
```

There is no need to go through the loop again, so use break.

Output:

```
One or more units passed? true
```

Continue with loops



continue example:

```
int passMark = 15;
int passesReqd = 3;
int[] score = { 4, 6, 2, 8, 12, 34, 9 };
for (int unitScore : score ) {
    if (unitScore < passMark) {
        continue;
    }
    passesReqd--;
    // Other processing
 }
 System.out.println("Units still reqd " + Math.max(0,passesReqd));
```

If unit failed, go on to check next unit.

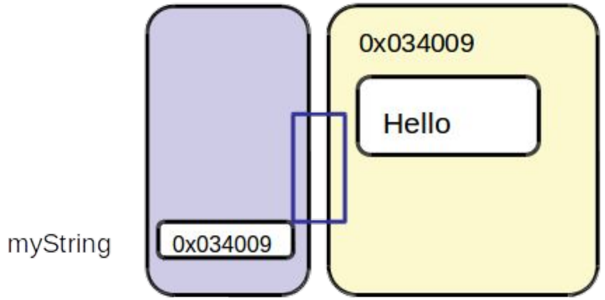
String Class

- The String class supports some non-standard syntax
- A String object can be instantiated without using the *new* keyword; in fact, this is the best practice:
    - String hisName = “John Smith";
- The new keyword can be used, but **it is not best practice**:
    - String herName = new String(“Mary Smith");
- Creating a String object using the new keyword may create one or two String objects in memory, while creating a String object by using a string literal may create none or one object;
- A String object is immutable; its value cannot be changed.
- A String object can be used with the string concatenation operator symbol (+) for concatenation.

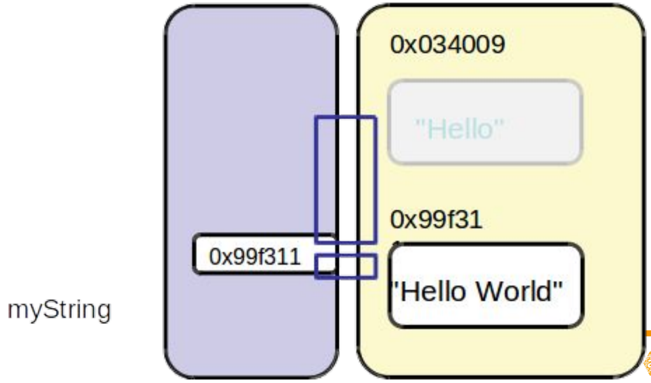
Concatenating Strings

- When you use a string literal in Java code, it is instantiated and becomes a String reference
- Concatenate strings:
  String name1 = "John"
  theirNames = name1 + " and " + "Mary Smith";
- The concatenation creates a new string, and the String reference theirNames now points to this new string.

```
String myString = "Hello";
```



```
String myString = "Hello";
myString = myString + " World";
```

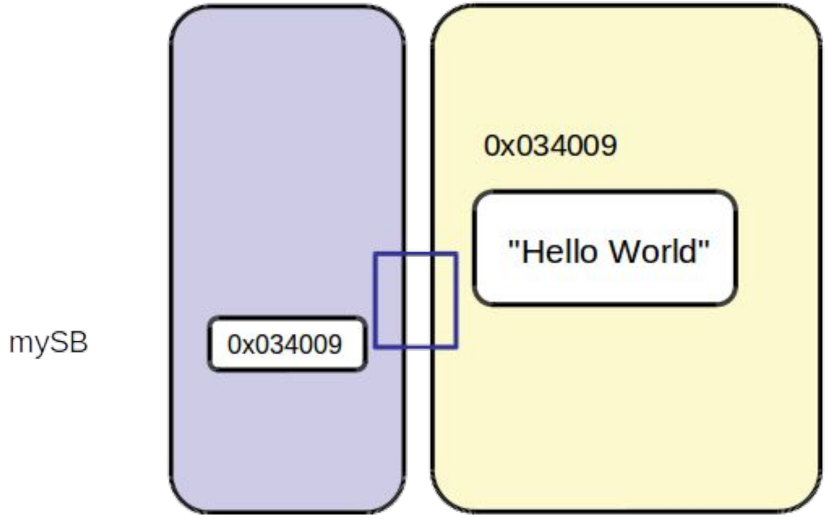


StringBuilder Class

- StringBuilder provides a mutable alternative to String
- StringBuilder:
  - Is a normal class. Use new to instantiate.
  - Has an extensive set of methods for append, insert, delete
  - Has many methods to return reference to current object.
  - Can be created with an initial capacity best suited to need
- String is still needed because:
  - It may be safer to use an immutable object
  - It has methods not available on StringBuilder:
  - startsWith
  - endsWith
  - ...

StringBuilder Class

```
StringBuilder mySB = new StringBuilder("Hello");
mySB.append(" World");
```



Java API Documentation

- Consists of a set of webpages;
- Lists all the classes in the API:
  - Descriptions of what the class does
  - List of constructors, methods, and fields for the class
- Highly hyperlinked to show the interconnections between classes and to facilitate lookup
- Available on the Oracle website at:
  - http://download.oracle.com/javase/8/docs/api/index.html

Luís Ribeiro

I’m a software engineer with more than 14 years of experience in software development in Java and other technologies, software architecture, team management and project management.

My mission is to transform people’s ideas into fully functional, production ready and user friendly software applications that can change the world!

luismmribeiro@gmail.com

Thank you