Java Classes

"Code with Passion!"

Topics

- Brief introduction on Object-Oriented Programming (OOP)
- Classes and objects
- Creation of Object instances using "*new*" keyword
- Methods: Instance methods vs. Static methods
- Variables (fields, properties)
- Scope of a variable

Brief Introduction on OOP

What is Object-Oriented Programming (OOP)?

- Revolves around the concept of objects as the basic elements of your programs
  - Object represent “things”
  - Object can be tangible things such as “Car”, “Computer” or intangible things such as “Course”, “Longevity”
- These objects are characterized by their properties (sometimes called attributes or states) and behaviors
- Key aspects of OOP
  - Encapsulation and Abstraction
  - Inheritance
  - Polymorphism

Example of Objects: Car and Lion
(The term Object here is a generic term)

Object	*Properties*	*Behavior*
Car	type of transmission manufacturer color	turning braking accelerating
Lion	Weight Color hungry or not hungry tamed or wild	roaring sleeping hunting

Classes and Objects (Object Instances)

What is a Class and Object Instances?

- Class
  - Represents a “type” from which object instances can be created
  - Can be thought of as a template, a boilerplate, a prototype or a blueprint of an object of same type
  - Is the fundamental structure in object-oriented programming

- What makes up a class?
  - Fields (they are also called properties or attributes) - specify the data types defined by the class
  - Methods - specify the behavior

Relationship between Class and Objects

- Object (or Object instance)
  - An object is an instance of a class
  - The property (field) values of an object instance is different from the ones of other object instances of a same class
  - Object instances of a same class share the same behavior (methods), however

Example: Classes and Objects

Car Class		Object Car A	Object Car B
Instance Variables	Plate Number	ABC 111	XYZ 123
	Color	Blue	Red
	Manufacturer	Mitsubishi	Toyota
	Current Speed	50 km/h	100 km/h
Instance Methods	Accelerate Method		
	Turn Method		
	Brake Method		

Example: Defining “Car” class

```
public class Car {

    // Fields - different values for different objects
    private String plateNumber;
    private String color;
    private String manufacturer;
    private int    speed;

    // Methods - common for all objects created from this class
    public void accelerate(){
        // Some code
    }

    public void turn(){
        // Some code
    }

    public void brake(){
        // Some code
    }
}
```

Classes and Reusability

- Classes provide the benefit of reusability
- Programmers create many object instances from the same class

What is Encapsulation?

- The scheme of hiding implementation details of a class
  - The user of the class does not need to know the implementation details of a class
  - The user can call *brake()* method of the *Car* class without knowing how the *brake()* method is actually implemented
- The implementation can change without affecting the user of the class

Creation of Object Instances with “new” keyword

How do you create Object Instance?

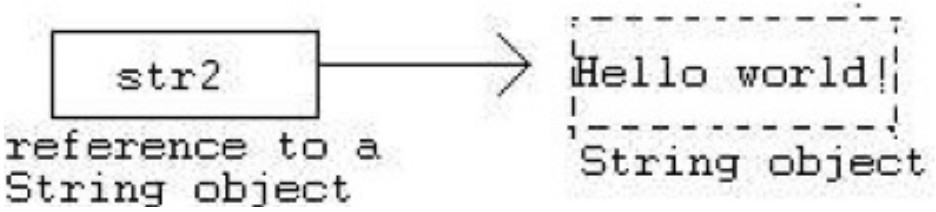
- To create an object instance of a class, use the new keyword
- For example, if you want to create an object instance of the class *String*, you would write the following code,

```
String str2 = new String("Hello world!");
```

or

```
String str2 = "Hello world!";
```

- String class is a special (and only) class you can create an instance without using new keyword as shown above



Constructor Method of a Class

- When you create an object using **`new`** keyword, the class' constructor method gets invoked automatically
  - Constructor method of a class typically contains some initialization logic

- Syntax of constructor method
  - The constructor method has the same name as the class
  - The constructor method does not have a return type
  - There could be multiple constructor methods (with different set of arguments – it is called constructor overloading)
  - If there is no constructor method, a no-arg constructor (sometimes called default constructor) gets inserted into the class by the compiler

Example: Constructor Method of Car Class

```
public class Car {

    // Fields - different values for different objects
    private String plateNumber;
    private String color;
    private String manufacturer;
    private int    speed;

    // Constructor method
    public Car() {
       // Some initialization can be done here
    }

    // Methods - common for all objects created from this class
    public void accelerate(){
        // Some code
    }

    public void turn(){
        // Some code
    }

    public void brake(){
        // Some code
    }
}
```

Lab:

Exercise 1: Create an Object Instance using “new” keyword
1011_javase_class_part1.zip

Methods (Instance methods & Static methods)

What is a Method?

- A method is a block of code (set of statements) that can be called to perform some specific task

- The following are characteristics of a method
  - It can return one or no values
  - It may accept as many arguments as possible it needs or no argument at all (Arguments are also called parameters)

Why Use Methods?

- Methods contain behavior of a class (business logic)
  - Taking a problem and breaking it into small, manageable tasks is critical to writing large programs.
  - We can do this in Java by a creating methods to perform these manageable tasks

There are Two Types of Methods

- Instance (non-static) methods
  - Can be called only through an object instance - so it can be called only after object instance is created
  - Calling syntax

    [NameOfObject].[methodName]
  - More common than static methods

- Static methods
  - Object instance does not have to be created
  - Can be called through a class

    [ClassName].[methodName]

Calling Instance (non-static) Methods

- To illustrate how to call methods, let's use the **`String`** class as an example

- You can use the Java API documentation to see all methods of the String class
  - http://docs.oracle.com/javase/8/docs/api/java/lang/String.html

- A method with “static” modifier is a static method while a method without “static” modifier is an instance (non-static) method

Calling Instance (non-static) Methods

- To call an instance method, we write the following,

```
nameOfObject.nameOfMethod( arguments );
```

```
// Create object instance of String class
String strInstance1 = new String("I am object
                      instance of a String class");

// Call charAt instance method of String class
char x = strInstance1.charAt(2);
```

Instance Methods

- Let's take two sample instance methods found in the String class

Method declaration	Definition
`public char charAt(int index)`	Returns the character at the specified index. An index ranges from 0 to length() - 1. The first character of the sequence is at index 0, the next at index 1, and so on, as for array indexing.
`public boolean equalsIgnoreCase (String anotherString)`	Compares this String to another String, ignoring case considerations. Two strings are considered equal ignoring case if they are of the same length, and corresponding characters in the two strings are equal ignoring case.

Example: Calling Instance Methods

```
// Create object instance of String class
String str1 = new String("HELLO");

// Call instance method charAt().
// This will return the character H
// and store it to variable x.
char x = str1.charAt(0);

// Create another object instance of String class
String str2 = new String("hello");

// Call instance method equalsIgnoreCase().
// This will return a boolean value true.
boolean  result = str1.equalsIgnoreCase( str2 );
```

Static Methods

- Static method definition
  - Static methods are defined with the keyword static
- Static method invocation
  - Static methods are invoked without creating an object instance (means without invoking the new keyword)
  - You call static method from a Class not object instance

```
Classname.staticMethodName(arguments);
```

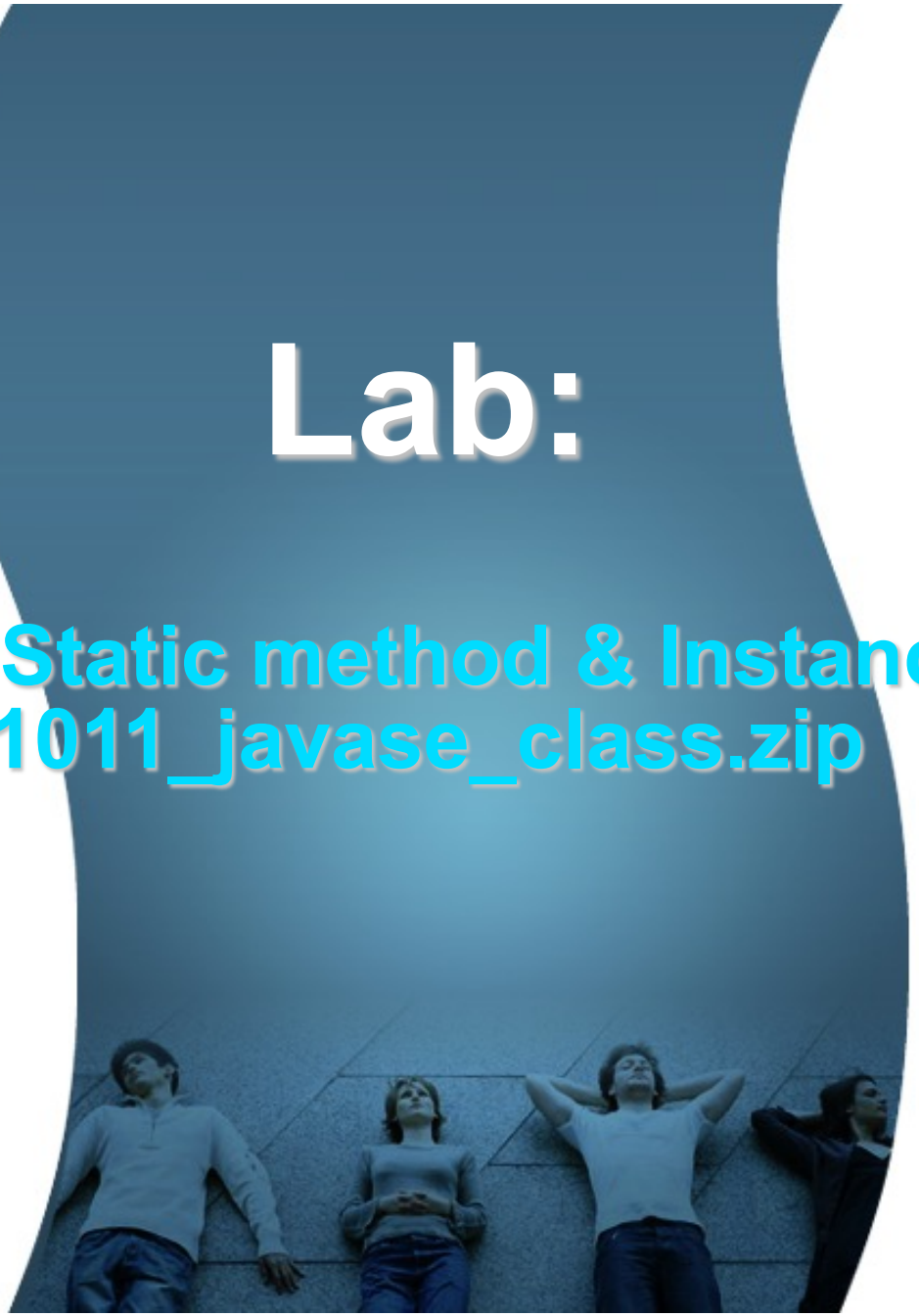
Static Method Invocation Example

```
// The parseInt() is a static method of the Integer class
// It converts the String 10, to an integer
int i = Integer.parseInt("10");

// The toHexString() is a static method of the Integer class.
// It returns a String representation of the integer
// argument as an unsigned integer base 16
String hexEquivalent = Integer.toHexString( 10 );
```

Lab:

Exercise 2: Static method & Instance Method
1011_javase_class.zip



Variables (Fields, Properties, Attributes)

Three Types of Variables

- There are three types of variables
  - Static variable (Also called as Class variable)
  - Non-static variable (Also called as Instance variable)
  - Local variable (Also called as automatic variable)

- The type of variable is determined by where the variable is declared

- The type of variable dictates where and how it can be used – this is called the scope of variable

Example: Types of Variables

```
public class Car {

    // Class (Static) variable
    private static String manufacturer = "Ford";

    // Instance (non-Static) variable
    private String plateNumber;
    private String color;

    public Car() {
    }

    public void accelerate(){
        // Local (automatic) variable
        int x = 10;
    }

}
```

Static Variable (Static Field)

- Declared inside a class body but outside of any method bodies (same as Instance variable)
- Prepended with the *static* modifier (different from Instance variable)
- Exists per each class
  - Come to existence when the class is loaded
- Shared by all object instances of the class

Instance Variable (Instance Field)

- Declared inside a class body but outside of any method bodies (like static variable)
- Exists per each object instance
  - Different object instances typically have different values for these instance variables
  - Come to existence when an object instance is created

Local Variable

- Declared within a method body
- Visible only within the method body
  - Come to existence only when the method gets executed

Scope of Variables

Scope of a Variable

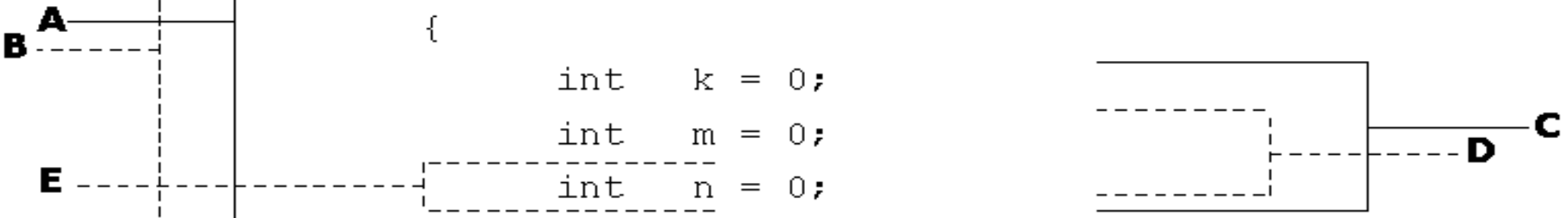
- The scope of a variable
  - Determines where in the program the variable is accessible.
  - Determines the lifetime of a variable or how long the variable can exist in memory

- The scope is determined by where the variable declaration is placed in the program
  - Just think of the scope as anything between the curly braces {...}, which represents a code block
  - More precisely, a variable's scope is inside the code block where it is declared, starting from the point where it is declared

Example 1: Scope of Variables

```
public class ScopeExample
{
    public static void main( String[] args ){
        int i = 0;
        int j = 0;

        //... some code here
        {
            int   k = 0;
            int   m = 0;
            int   n = 0;
        }
}
```

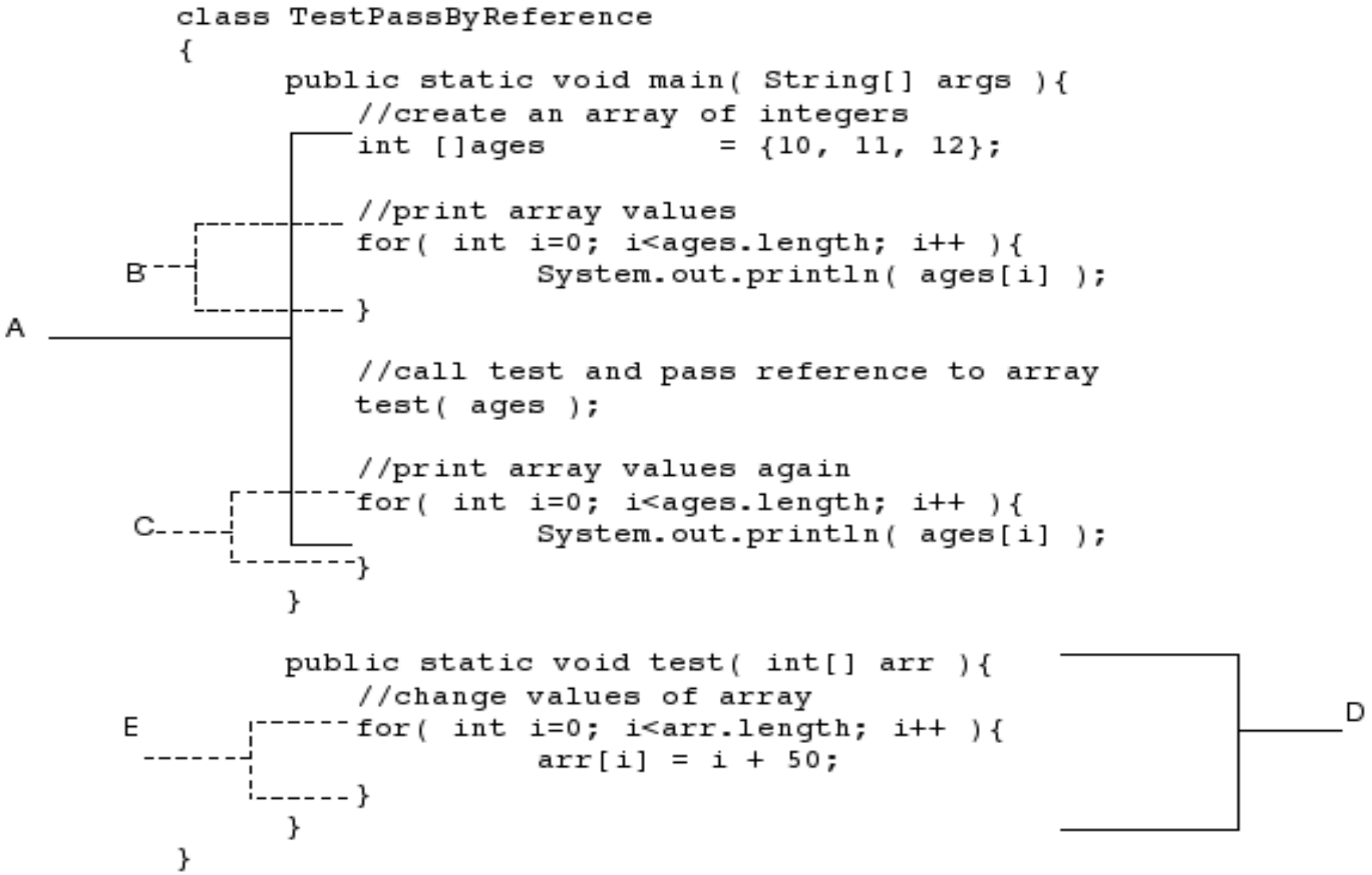


Example 1: Explanation

- The code we have in the previous slide represents five scopes indicated by the lines and the letters representing the scope

- Given the variables i,j,k,m and n, and the five scopes A,B,C,D and E, we have the following scopes for each variable:
  - The scope of variable i is A.
  - The scope of variable j is B.
  - The scope of variable k is C.
  - The scope of variable m is D.
  - The scope of variable n is E.

Example 2: Scope of Variables



Example 2: Explanation

- In the main method, the scopes of the variables are,
  - ages[] - scope A
  - i in B - scope B
  - i in C – scope C
- In the test method, the scopes of the variables are,
  - arr[] - scope D
  - i in E - scope E

Scope of a Variable

- When declaring variables, only one variable with a given identifier or name can be declared in a scope.
- That means that if you have the following declaration,

```
{
        int test = 10;
        int test = 20;
}
```

This will cause a compile error since names have to be unique within a block

Scope of a Variable

- However, you can have two variables of the same name, if they are declared in different blocks. For example,

```
public class Main {
    static int test = 10;

    public static void main(String[] args) {

        System.out.println(test);   // prints 10

        // test variable is defined in a new block
        {
            int test = 20;
            System.out.println(test);// prints 20
        }

        System.out.println(test);   // prints 10
    }
}
```

Scope of Variables

- Local (automatic) variable
  - Only valid from the line they are declared on until the closing curly brace of the method or code block within which they are declared
  - Most limited scope
- Instance variable
  - Valid as long as the object instance is alive
- Class (static) variable
  - In scope from the point the class is loaded into the JVM until the the class is unloaded
  - Class are loaded into the JVM the first time the class is referenced

Lab:

Exercise 3: Scope of Variables
1011_javase_class.zip

Code with Passion!