

# Test Automation Engineering Fundamentals: Java

## Module 4: Classes and Objects

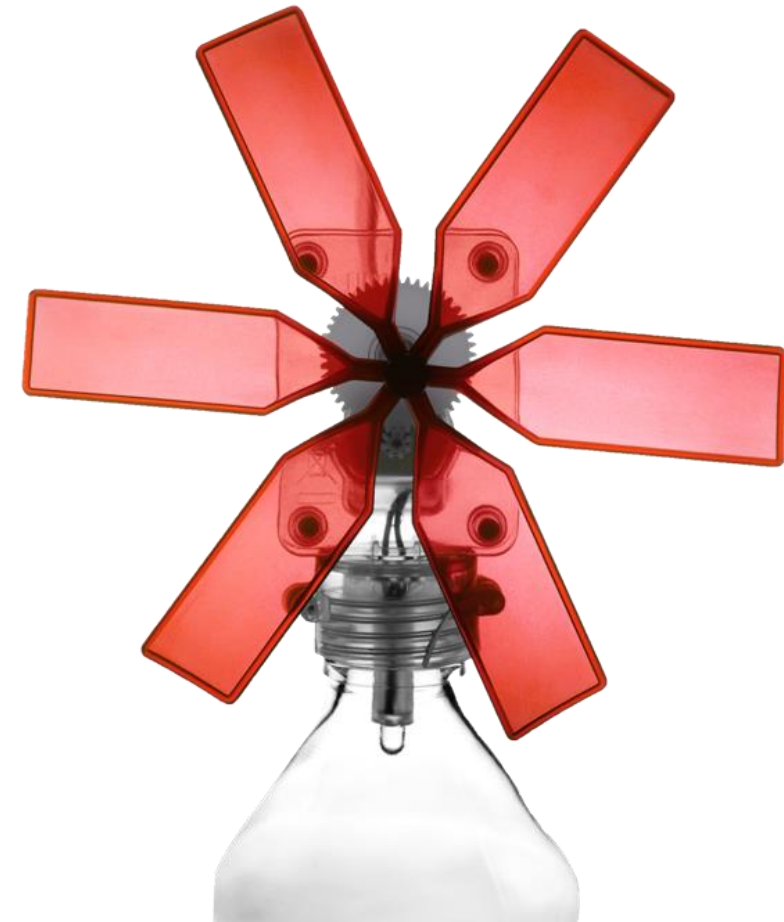
**GROW  
CONFI  
DENTLY**

## **MIT Faculty Video: “Classes and Objects”**

# Module Objectives

**At the end of this module, participants will be able to:**

- Define and apply the concept of Object-Oriented Programming
- Define and differentiate classes from instances
- Define the concept of Garbage Collection
- Explain the Java package structure and class path
- Create, use, and import classes
- Define the concept of a constructor and finalizer
- Identify and use access modifiers
- Identify and create static attributes and methods

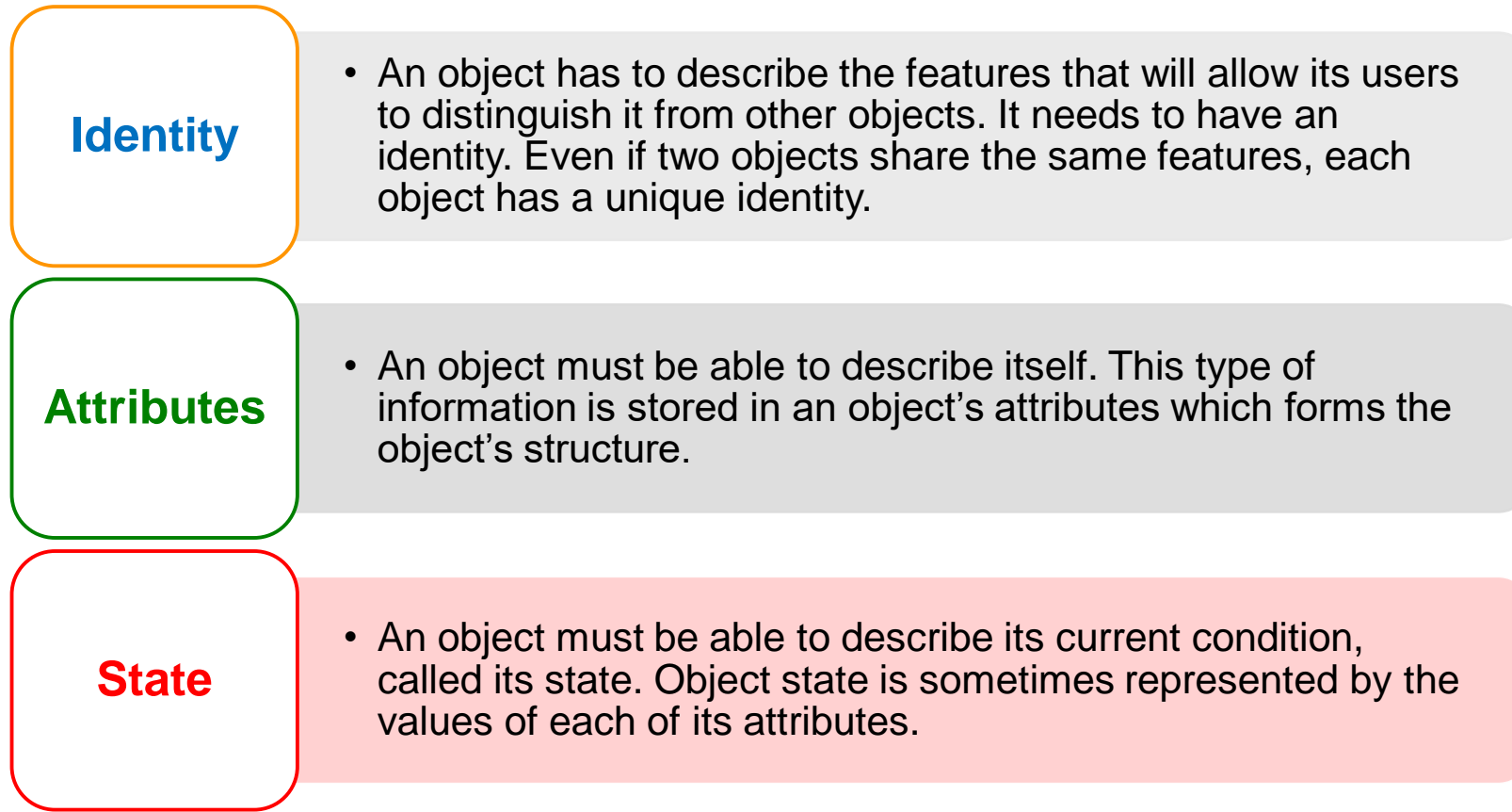


# Overview of Object-Oriented Programming

- Object-Oriented Programming is a programming pattern that makes use of objects and their interactions to design and implement applications
- Objects are entities that serve as the basic building blocks of an object-oriented application
- An object is a self-contained entity with attributes and behaviors

# Information Within an Object

Each object defines three basic types of information:



# Classes vs. Instances

- A *class* is a template for a specific object.
- A *class* defines the attributes and methods that all objects belonging to the class have.
- The attributes and methods of a class are called 'fields' or 'members'.
- An *instance* refers to an object that is a member of a particular class. All objects that belong to a class are instances of that class.



# Classes vs. Instances (cont.)

Car  
=  
Class

Your car  
=  
Instance



# Creating and Manipulating Instances of a Class

- Instances of a class can be created by using the keyword **new**. This process is called 'instantiation'.
- The 'new' keyword creates the object based on the specified class and returns a reference to the newly created object.
- The reference is then received by an appropriate identifier.

```
<Class Type> <Identifier> = new <Class Type>(<Constructor Parameters>)
```

```
Person him = new Person("John Doe", 25, 'M');
```



Refer to the ClassInstanceSample.java sample code.



# Creating and Manipulating Instances of a Class (cont.)

- Fields of object instances can be accessed through their identifiers names.
- Access to member fields and methods is done through the use of the 'dot' notation.

```
<reference>.<attribute name>  
<reference>.<member method>(<parameters>);
```

```
him.setAge(23)  
him.getName();
```



Refer to the ClassInstanceSample.java and Person.java sample code.

# Custom Classes

- A primary aspect of an Object-Oriented programming language is the ability to define customized classes.
- A class declaration is the basic building block of a Java application.
- All code is built and attached to a class declaration.  
There is no code that does not belong to a class.

# Custom Classes (cont.)

- A Java Class denotes a category of objects, and acts as a blueprint for creating such objects.
- Classes declare what *package* they belong to.
- Classes declare what classes to *import* from other packages.
- Fields (also known as *variables* or *attributes*) refer to the properties of the class.
- Methods (also known as *operations*) refer to behaviors that the class exhibits.
- The source file containing the class declaration must have the same name as its public class.

# Package Structure and the Class Path

- Packages are collections of classes.
- Packages are used to group classes together based on functionality or relationship.
- Classes can be assigned to a certain package by using the *package* keyword:

```
package <package_name>;  
package sef.module4.samples;
```

- The package structure is representative of the directory structure where the class definition is located.



# Package Structure and the Class Path (cont.)

- Class path is a list of directories and JAR files that contain classes and other resources needed by a Java application to run.
- Classes not included in the class path cannot be used by Java during compiling, runtime, and other utilities.
- A class path can be set in many ways, but a common way involves setting the CLASSPATH environment variable in the command prompt.
- A class path can contain more than one directory path.

```
C:> set CLASSPATH=classpath1;classpath2...;  
C:> set CLASSPATH=.;C:\SEF;
```

# Importing Classes

- A program can use external resources by importing it.
- Imported resources can be a single class or an entire package of classes
- The imported classes need to be visible in the class path in order to be accessible.

```
import <package_name>; OR import <class-name>;  
import java.util.*;  
import java.util.Calendar;
```

# Activity 1 – Create a Java File

- In this activity, you will:
  - Create a java file DateActivity.java inside the package sef.module4.activity
  - Import java.util.Date
  - Create instance d of Date using Date()
  - Print today's date



# Garbage Collection

- The keyword 'new' explicitly allocates resources for an object, but there is no explicit way of telling the system to 'de-allocate' an object.
- Garbage collection is the process of automatically clearing up the memory being used by objects that will no longer be used by the program.
- Invocation of the garbage collector is automatically determined by the Java Virtual Machine, depending on the need to do so.
- An application can 'suggest' that the system initiate garbage collection using *System.gc()* or *Runtime.getRuntime().gc()*.



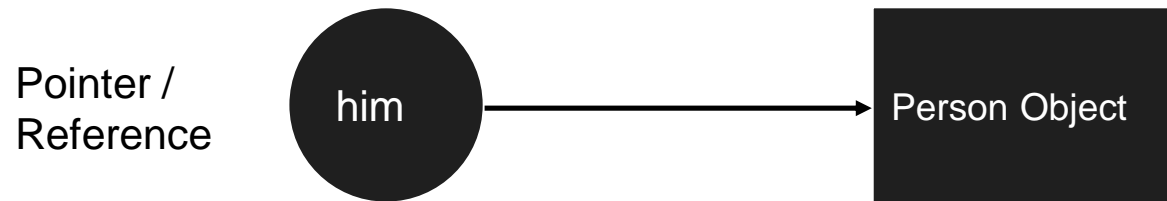
# Garbage Collection (cont.)

- Garbage collection is used by Java to:
  - Recover resources (particularly memory) that are much needed by an application.
  - Avoid dangling pointers.
  - Ensure program integrity.
- Dangling pointers are the occurrence of having variables or references pointing to objects in memory that are not of the appropriate type or value.

# Garbage Collection (cont.)

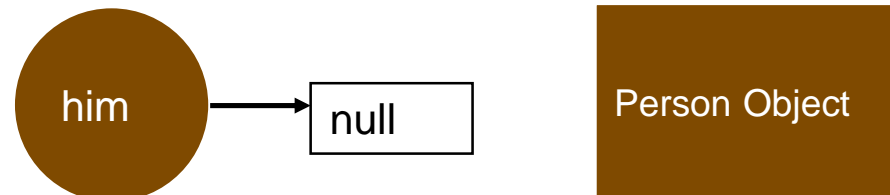
- Objects are safe from garbage collection if there is at least one reference variable that refers to the object

```
Person him = new Person();
```



- Objects are eligible for garbage collection when there are no more identifiers that refer to the object

```
him = null;
```



# Concept of Constructor

- Constructors are methods that set the initial state of an object
- Constructor name must be the same as the class name
- Constructor cannot return a value, not even **void**
- Constructors cannot be inherited
- If the class does not explicitly declare a constructor, it will default to a no parameter, do-nothing constructor

```
public class Cubel
{
    private int size;

    Cubel(int size)
    {
        this.size = size;
    }
}
```

# Concept of Finalizer

- A finalizer is a special method that is executed to remove the object or resource from memory when it is no longer needed.
- The finalize() method is used by the garbage collector to free up memory from resources which are no longer needed.

```
protected void finalize() throws Throwable
{
    try
    {
        close(); // close open files
    }
    finally
    {
        super.finalize();
    }
}
```

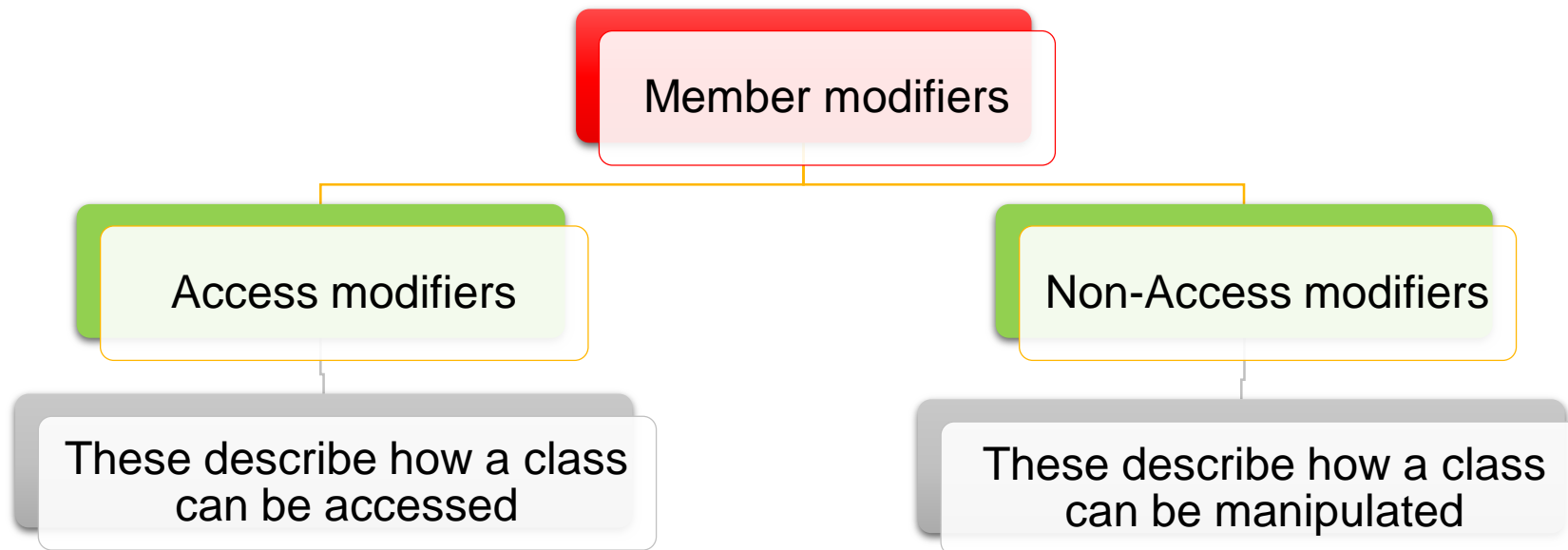


# Attributes and Behavior

- Variables declared as part of the class represent the data that is contained by instances of this class
- The values stored by these variables represent the object's current 'attributes' or 'state'
- Methods declared as part of the class represent the operations that instances of the class can perform
- These operations are said to be the 'behavior' of the object

# Member Modifiers

- Classes often need control on how their attributes and behaviors are accessed.
- Member modifiers allow this control.
- There are two types of member modifiers.



# Access Modifiers

- Access modifiers describes how a class can be accessed.
- There are four types of access modifiers. These are:

**default /  
no modifier**

- A class can be accessed by classes belonging to the same package

**public**

- A class member can be accessed by any class in any package

**protected**

- A class member can only be accessed by itself or its subclasses

**private**

- A class member can only be accessed by itself

# Non-access Modifiers

- Non-access modifiers - describes how a class can be manipulated.

**static**

- Member belonging to a class; shared by all instances of the class

**final**

- Member declared as a constant. It cannot be modified once declared

**abstract**

- Method is declared but is never instantiated. This modifier can be applied to methods to extend a class

**strictfp**

- Method implements strict floating-point arithmetic



# Non-access Modifiers (cont.)

- Non-access Modifiers - describes how a class can be manipulated.

**synchronized**

- Method is executed by only one thread at a time. It can be applied to blocks of code and methods

**native**

- Method implementation is written in other language. It can only be applied to methods.

**transient**

- An instance variable is not saved when its object is persisted or serialized. It can only be applied to variables

**volatile**

- Variable is modified asynchronously by concurrently running threads. It can be applied only to variables

# Restrictions on Use of Modifiers

- Not all combinations of instance and class variables and methods are allowed:
  - Instance methods can access instance variables and instance methods directly.
  - Instance methods can access class variables and class methods directly.
  - Class methods can access class variables and class methods directly.
  - Class methods cannot access instance variables or instance methods directly. Class methods also cannot use the `this` keyword as there is no instance for this to refer to.

# Static Attributes and Methods

- Static attributes and methods are associated to the class and are shared by all instances of the class
- Static attributes and methods are defined by using the keyword **static**
- Only static methods can directly access attributes and call methods that are also static
- Static members of a class can be accessed or are referenced by the class name

```
<Class Name>.<static member>  
Calendar.getInstance();
```

# Static Code Blocks

- Code blocks can be marked off as static
- Static code blocks are executed only once when the class is loaded by the JVM for the first time
- Only static code blocks can directly access static attributes and methods

```
public class SampleClass{  
  
    static{  
        //your code here  
    }  
  
}
```



Refer to the StaticSample2.java sample code.

# Activity 2 – Create a Calculator

- In this activity, you will:
  - Open the file 'Calculator.java' in the package sef.module4.activity
  - Read the instructions and create the code to complete this program



# Questions and Comments

- What questions or comments do you have?

