Introducción a Java

Java 21

JoeDayz | 2025

Course Goals

In this course, you learn how to implement application logic using Java SE:

- Describe the object-oriented programming approach
- Explain Java syntax and coding conventions
- Use Java constructs and operators
- Use core Java APIs, such as Collections, Streams, I/O, and Concurrency
- Deploy Java SE applications

Audience

The target audience includes those who:

- Have some non-Java programming experience and want to learn Java
- Have basic knowledge of Java and want to improve it
- Prepare for the Java SE 21 Certification exam

Course Structure

Lesson 1: Introduction to Java
Lesson 2: Primitive Types, Operators, and Flow Control Statements
Lesson 3: Text, Date, Time, and Numeric Objects

Lesson 4: Classes and Objects
Lesson 5: Improved Class Design
Lesson 6: Implement Inheritance and Use Records

Lesson 7: Interfaces and Generics
Lesson 8: Arrays and Loops
Lesson 9: Collections

Lesson 10: Nested Classes and Lambda Expressions
Lesson 11: Java Streams API
Lesson 12: Exception Handling, Logging, and Debugging

Lesson 13: Java I/O API
Lesson 14: Java Concurrency and Multithreading
Lesson 15: Modules and Deployment

Extras:

Appendix A:
Annotations

Appendix B:
Java Database Connectivity

Appendix C:
Java Security

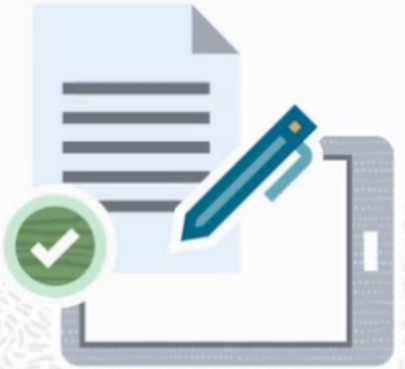
Appendix D:
Advanced Generics

Appendix E:
Java Applications on Oracle Cloud

Appendix F:
Miscellaneous Java Topics

Course Practices

- During the course practice sessions, you:
  - Explore the features of Java language
  - Apply the knowledge gained throughout the course to develop a product management application
- The practice environment uses:
  - JDK 21
  - JShell
  - IntelliJ 2023.2.4



Java Features y Object Oriented Concepts

Objectives

After completing this lesson, you should be able to:

- Discover Java language origins and use cases
- Explain Java portability and provider neutrality
- Explain object-oriented concepts
- Describe Java syntax and coding conventions
- Create a Java class with main method
- Compile and execute a Java application

What Is Java?

- It is a general-purpose programming language similar to C and C++.
- It is object oriented and platform independent.
- It was originally designed in 1995 for use in consumer electronics.
- Modern uses include writing applications for Internet of Things, cloud computing, and so on.
- This course covers Java Standard Edition (SE) version 21.

- ***Java Editions:***

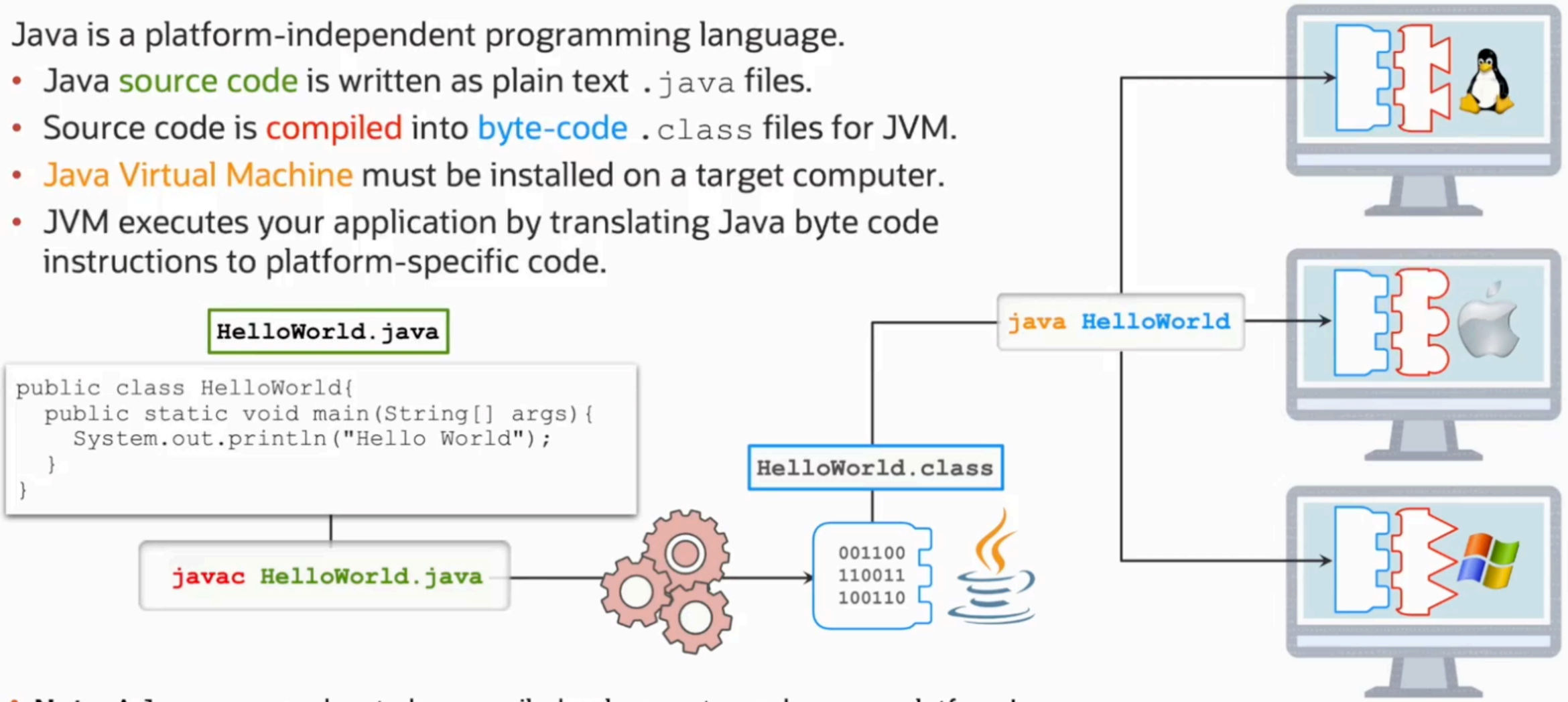
  Java Card - Smart Card Edition
  Java ME - Micro Edition
  Java SE - Standard Edition
  Java MP - Micro Profile
  Jakarta EE - (Java EE) Enterprise Edition

- ***Java SE is the base edition on which other editions are based.***

How Java Works

Java is a platform-independent programming language.

- Java source code is written as plain text `.java` files.
- Source code is compiled into byte-code `.class` files for JVM.
- Java Virtual Machine must be installed on a target computer.
- JVM executes your application by translating Java byte code instructions to platform-specific code.



✣ **Note:** A Java program has to be compiled only once to work on any platform!

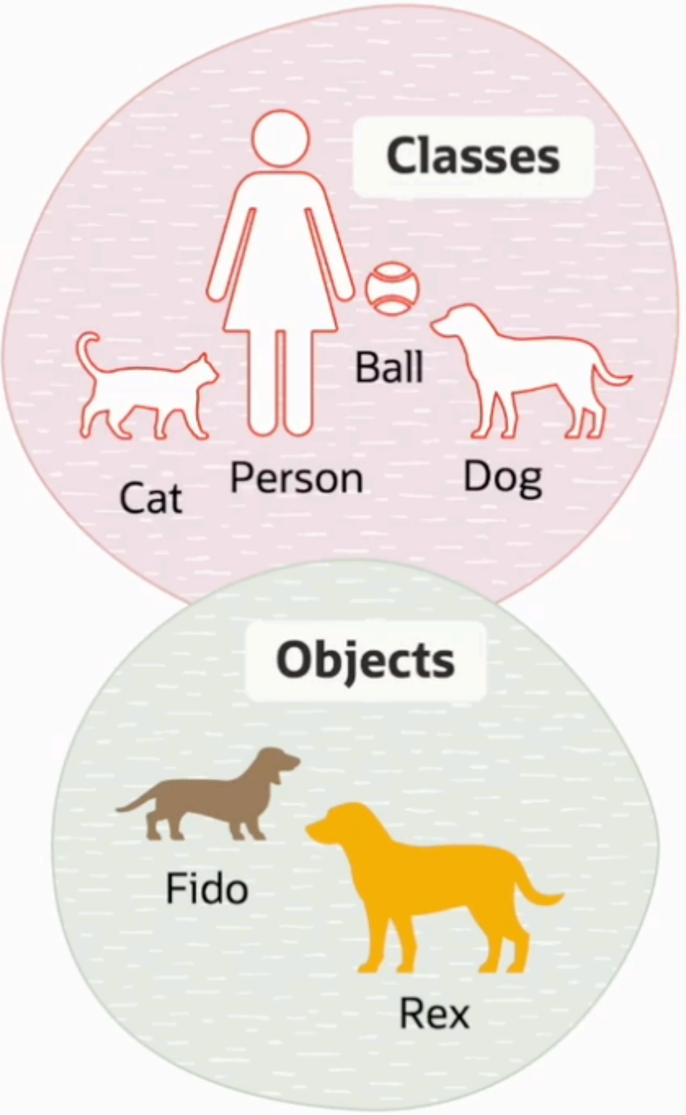
Object-Oriented Principles

- **Abstraction:** The software objects represent a subset of behaviors and information of the real-world objects.
- **Encapsulation:** The software object hides its implementation details and internal state.
- **Inheritance:** A specialized type (child) inherits code from the more general type (parent).
- **Polymorphism:** Objects of different types respond differently to the same message.

Classes and Objects

Class and object are two key object-oriented concepts.

- Java code is structured with classes.
- Class represents a type of thing or a concept, such as: Dog, Cat, Ball, Person.
- Class is an abstraction of a real-world thing or concept.
- An object is a specific instance (example of) a class.




```
class Person {
  void play() {
    Dog dog = new Dog();
    dog.name = "Rex";
    Ball ball = new Ball();
    dog.fetch(ball);
  }
}
```

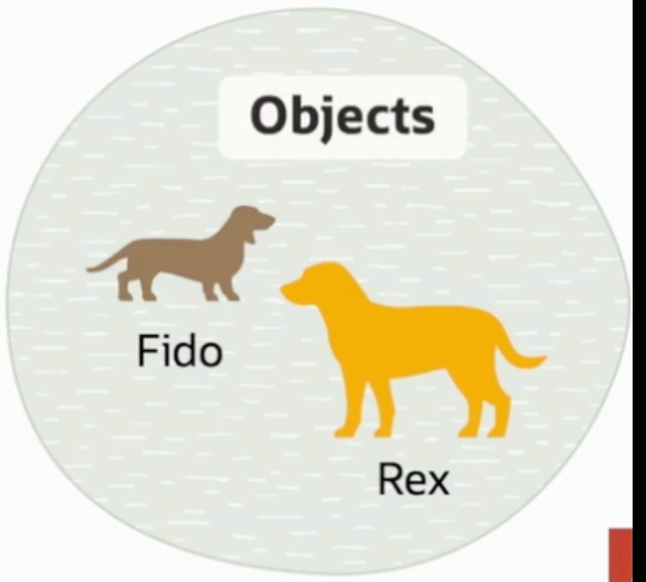
```
class Dog {
  String name;
  fetch(Ball ball) {
    ball.find();
    ball.chew();
  }
}
```

Objects

- Each object would be capable of having specific values for each attribute defined by a class that represents its type.
- Examples:
  - A dog could be called Fido, and be brown and small.
  - Another could be called Rex, and be orange and big.
- To operate on an object, you can reference it by using a variable of a relevant type.
- Each object would be capable of behaviors defined by a class that represents its type:
  - At run time, objects invoke operations upon each other to execute program logic.

```
class Person {
  void play() {
    Dog dog = new Dog();
    dog.name = "Rex";
    Ball ball = new Ball();
    dog.fetch(ball);
  }
}
```

```
class Dog {
  String name;
  fetch(Ball ball) {
    ball.find();
    ball.chew();
  }
}
```



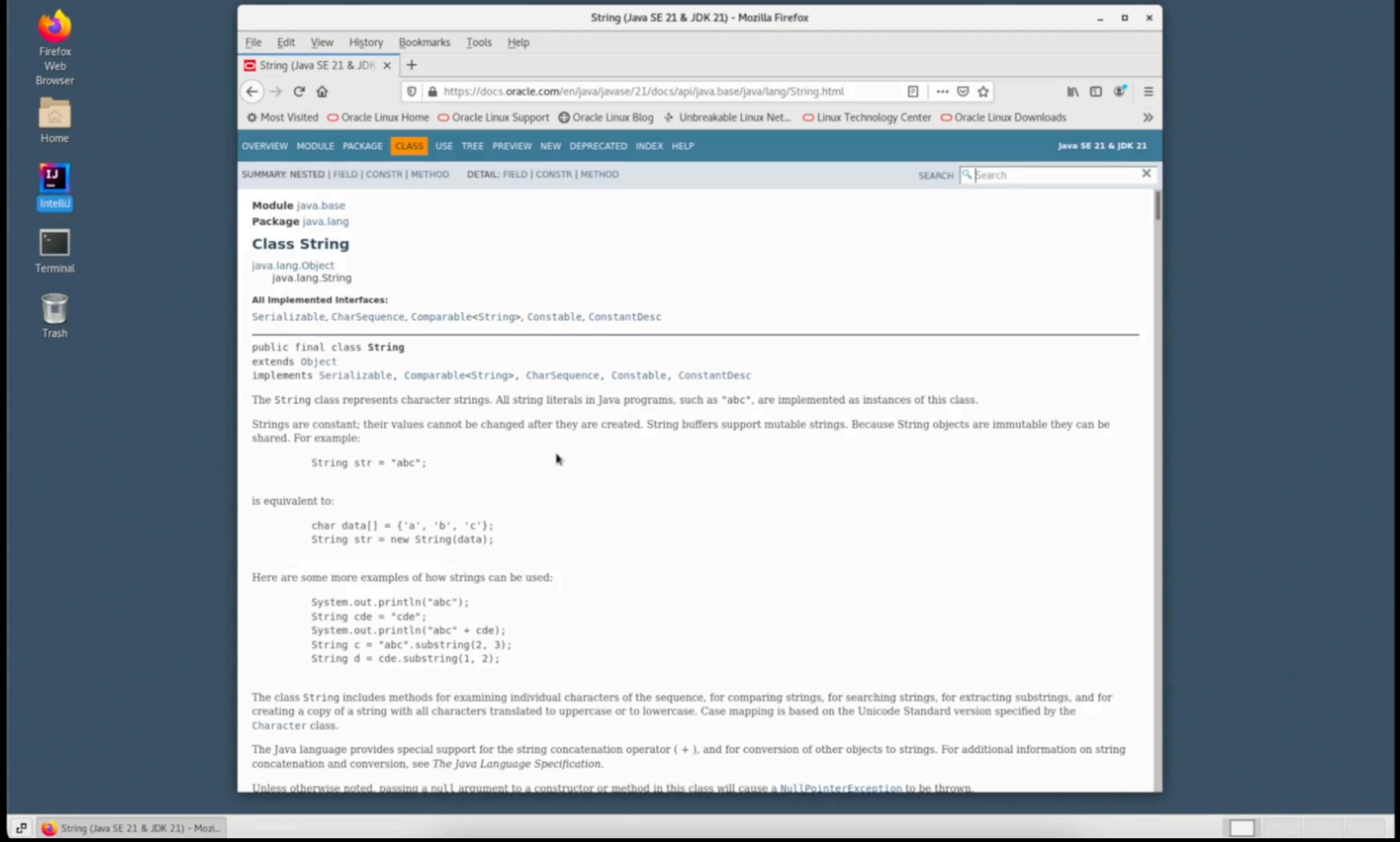
Java APIs

Java Development Kit (JDK) provides hundreds of classes for various programming purposes:

- To represent basic data types, for example, `String`, `LocalDateTime`, `BigDecimal`, and so on
- To manipulate collections, for example, `Enumeration`, `ArrayList`, `HashMap`, and so on
- To handle generic behaviors and perform system actions, for example, `System`, `Object`, `Class`, and so on
- To perform input/output (I/O) operations, for example, `FileInputStream`, `FileOutputStream`, and so on

Many other API classes are used to access databases, manage concurrency, enable network communications, execute scripts, manage transactions, security, and logging, build graphical user interfaces, and so on.

- Application Programming Interface (API) is a term that describes a collection of classes that are designed to serve a common purpose.
- All Java APIs are thoroughly documented for each version of the language. Java SE documentation can be found at: https://docs.oracle.com/en/java/javase/index.html


Applications
Places
Firefox
Fri Mar 29, 19:35
Firefox Web Browser
Home
IntelliJ
Terminal
Trash
String (Java SE 21 & JDK 21) - Mozilla Firefox
File Edit View History Bookmarks Tools Help
String (Java SE 21 & JDK
https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/String.html
Most Visited
Oracle Linux Home
Oracle Linux Support
Oracle Linux Blog
Unbreakable Linux Net...
Linux Technology Center
Oracle Linux Downloads
OVERVIEW MODULE PACKAGE CLASS USE TREE PREVIEW NEW DEPRECATED INDEX HELP
Java SE 21 & JDK 21
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD
SEARCH
Search
Module java.base
Package java.lang
Class String
java.lang.Object
java.lang.String
All Implemented Interfaces:
Serializable, CharSequence, Comparable<String>, Constable, ConstantDesc
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence, Constable, ConstantDesc
The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.
Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:
String str = "abc";
is equivalent to:
char data[] = {'a', 'b', 'c'};
String str = new String(data);
Here are some more examples of how strings can be used:
System.out.println("abc");
String cde = "cde";
System.out.println("abc" + cde);
String c = "abc".substring(2, 3);
String d = cde.substring(1, 2);
The class String includes methods for examining individual characters of the sequence, for comparing strings, for searching strings, for extracting substrings, and for creating a copy of a string with all characters translated to uppercase or to lowercase. Case mapping is based on the Unicode Standard version specified by the Character class.
The Java language provides special support for the string concatenation operator (+), and for conversion of other objects to strings. For additional information on string concatenation and conversion, see The Java Language Specification.
Unless otherwise noted, passing a null argument to a constructor or method in this class will cause a NullPointerException to be thrown.
String (Java SE 21 & JDK 21) - Mozi...

Java Naming Conventions

- Java is case-sensitive; `Dog` is not the same as `dog`.
- Package name is a reverse of your company domain name, plus the naming system adopted within your company.
- Class name should be a noun, in mixed case with the first letter of each word capitalized.
- Variable name should be in mixed case starting with a lowercase letter; further words start with capital letters.
- Names should not start with numeric characters (`0-9`), underscore (`_`) or dollar (`$`) symbols.
- Constant name is typically written in uppercase with underscore symbols between words.
- Method name should be a verb, in mixed case starting with a lowercase letter; further words start with capital letters.

```
package:   com.oracle.demos.animals
class:     ShepherdDog
variable:  shepherdDog
constant:  MIN_SIZE
method:    giveMePaw
```

```
package:   animals
class:     Shepherd Dog
variable:  _price
constant:  minSize
method:    xyz
```

- **Note:** The use of the _ symbol as a first or only character in a variable name produces a compiler warning in Java 8 and an error in Java 9 onwards.

Java Basic Syntax Rules

- All Java statements must be terminated with the " ; " symbol.
- Code blocks must be enclosed with " { " and " } " symbols.
- Indentations and spaces help readability, but are syntactically irrelevant.

```
package com.oracle.demos.animals;
class Dog {
  void fetch() {
    while (ball == null) {
       keepLooking();
    }
  }
  void makeNoise() {
    if (ball != null) {
      dropBall();
    } else {
      bark();
    }
  }
}
```

✣ **Note:** The example shows some constructs such as `if/else` and `while` that are covered later in the course.

Define, Compile and Execute a Java Program

Defining a Java Class

- Class name is typically represented by one or more nouns: `Dog`, `GreatCat`.
- Class must be saved into a file with the same name and the `.java` extension.
- Classes are grouped into packages, represented as folders where class files are saved.
- Package name is typically a reverse of your company domain name, plus a naming system adopted within your company: `com.oracle.demos`, `org.acme.something`.
- Package and class name must form a unique combination.

```
package <package name>;
class <ClassName> {
}
```

```
/somepath/com/oracle/demos/animals/Dog.java
```

```
package com.oracle.demos.animals;
class Dog {
   // the rest of this class code
}
```

Note: If package definition is missing, class would belong to a "default" package and would not be placed into any package folder. However, this is not a recommended practice.

Accessing Classes Across Packages

To access a class in another package, do one of the following:

- Prefix the class name with the package name.
- Use the import statement to import specific classes or the entire package content.
  - The import of all classes from the `java.lang.*` package is implicitly assumed.

The example shows three alternative ways of referencing the class `Dog` in the package `animals` from the class `Owner` in the package `people`:

```
package animals;
public class Dog {
}
```

```
package people;
public class Owner {
   animals.Dog myDog;
}
```

```
package people;
import animals.Dog;
public class Owner {
   Dog myDog;
}
```

```
package people;
import animals.*;
public class Owner {
   Dog myDog;
}
```

Notes

- Imports are not present in a compiled code. An import statement has no effect on the runtime efficiency of the class. It is a simple convenience to avoid prefixing class name with package name throughout your source code.
- Access modifiers (such as public) are explained in the following slide.

Implementing Encapsulation with Access Modifiers

Access modifiers describe the visibility of classes, variables, and methods.

- `public`: Visible to any other class
- `protected`: Visible to classes that are in the same package or to subclasses
- no access modifier (default): Visible only to classes in the same package
- private: Visible only within the same class

```
package <package name>;
import <package name>.<class name>;
import <package name>.*;
<access modifier> class <ClassName> {
  <access modifier> <variable definition>
  <access modifier> <method definition>
}
```

```
package b;
public class Y
        extends a.X {
  public void doThings() {
    Y p = new Y();
    p.m1="Hello";
    p.m2="Hello";
    p.m3="Hello";
    p.m4="Hello";
  }
}
```

```
package a;
public class X {
  public String m1;
  protected String m2;
  String m3;
  private String m4;
}
```

Notes:

- Subclass-superclass relationship (use of the `extends` keyword) is covered later in the course.
- Any nonprivate parts of your class should be kept as stable as possible, because changes to such code may adversely affect several other classes that may be using your code.

Creating a Main Application Class

The main method is the entry point into your application.

- It is the starting point of program execution.
- The method must be called main.
- It must be public. You intend to invoke this method from outside this class.
- It must be static. Such methods can be invoked without creating an instance of this class.
- It must be void. It does not return a value.
- It must accept array of String objects as the only parameter.

(The name of this parameter "`args`" is irrelevant.)

```
package demos;
public class Whatever {
  public static void main(String[] args) {
    // program execution starts here
  }
}
```

✣ **Note:** Use of static and void keywords and handling of arrays are covered later in the course.

Compiling a Java Program

Compile classes with the javac Java compiler.

- The `-classpath` or `-cp` parameter points to locations of other classes that may be required to compile your code.
- The `-d` parameter points to a path to store the compilation result.

  (The compiler creates package subfolders with compiled class files in this path.)
- Provide path to source code.

```
javac -cp /project/classes -d /project/classes /project/sources/demos/Whatever.java
```

```
package demos;
public class Whatever {
  public static void main(String[] args) {
    // program execution starts here
  }
}
```


```
/project/classes/demos/Whatever.class
```

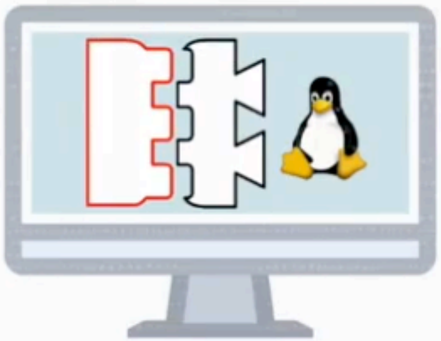
Executing a Java Program

Execute a program using the **`java`** executable Java Virtual Machine (JVM).

- Specify `-classpath` or `-cp` to point to folders where your classes are located.
- Specify fully qualified class name. Use a package prefix; do not use the `.class` extension.
- Provide a space-separated list of parameters after the class name.

Access command-line parameters:

- Use an array object to access parameters.
- Array index starts at 0 (first parameter).



`java`

```
package demos;
public class Whatever {
  public static void main(String[] args) {
    String param1 = args[1];
    System.out.println("Hello "+param1);
  }
}
```

```
java  -cp /project/classes demos.Whatever Jo John "A Name" Jane
Hello John
```

- Since Java 11, it is also possible to run single-file source code as if it is a compiled class.
  JVM will interpret your code, but no compiled class file would be created:

```
java /project/sources/demos/Whatever.java
```

Comments and Documentation

- Code Comments can be placed anywhere in your source code.
- Documentation Comments:
  - May contain HTML markups
  - May contain descriptive tags prefixed with the @ sign
  - Are used by the Javadoc tool to generate documentation

Package demos

Class Whatever

java.lang.Object
    demos.Whatever

public class **Whatever**
extends Object

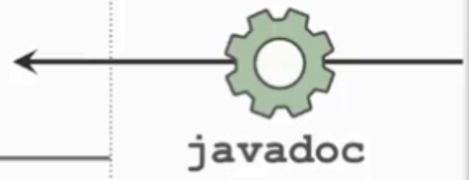
The Whatever class represents an example of documentation comment

Version:

1.0

Author:

oracle




```
// single-line comment

/*
   multi-line comment
*/

/**
*  The {@code Whatever} class
*  represents an example of
*  documentation comment
* @version 1.0
* @author oracle
*/
```

```
javadoc -d <documentation path>
        -sourcepath <source code path>
        -subpackages <name of the root package>
```

Note: All APIs in the Java Development Kit are documented using the Javadoc utility.

Code Snippets in Javadoc

- A `@snippet` tag simplifies the inclusion of example source code in documentation.
- Snippet features include highlighting, replacing, and linking code fragments.

```
/**
 * The following code shows how to use{@code Purchase.isComplete}:
 * {@snippet :
 * if (!p.isComplete()) {
 * LocalDate d; // @link substring="LocalDate" target="java.time.LocalDate"
 * d = p.completeNow(); // @highlight substring="completeNow"
 * }
 * }
*/
```

✣**Note:** Code snippets were introduced in Java SE 18.

External Snippets

- An external snippet refers to a separate file that contains the content of the snippet.
- Snippets may be used in various files, such as Java sources, property bundles.

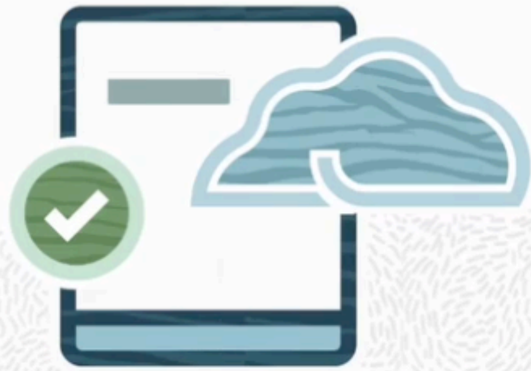
```
/**
 * The following code shows how to use{@code Purchase.isComplete}:
 * {@snippet class="Shop.java" region="example"}
*/
```

```
public class Shop {
  void buyProducts(Purchase p) {
    // @start region="example"
    if (!p.isComplete()) {
      p.completeNow();
    }
    // @end
  }
}
```

Summary

In this lesson, you should have learned how to:

- Discover Java language origins and use cases
- Explain Java portability and provider neutrality
- Explain object-oriented concepts
- Describe Java syntax and coding conventions
- Create a Java class by using the main method
- Compile and execute a Java application



Practica

Quiz

1. Java uses ____ to structure the code and ____ for individual instances.

- ○ Objects, Classes
- ○ Classes, Objects
- ○ Classes, Attributes
- ○ Classes, References

2. Which is an invalid variable name?

- ○ $_exceptionValue
- ○ 4ScoreAnd8Years
- ○ _SystemValue
- ○ MyNewValue
- ○ object

3. Which two statements are true?

- [ ] Java source code must be compiled into classes for each platform.
- [ ] Java is a specialized programming language for use in browsers.
- [ ] Java source code is plain text.
- [ ] Java is an object-oriented programming language.

4. Source code is saved as ____ files and code is compiled into ____ files.

- ○ .java, .exe
- ○ .java, .bytes
- ○ .java, .class
- ○ .text, .java