

Conceptos fundamentales de Java SE 7

Volumen I - Guía del Alumno

D67234CS20

Edición 2.0

Noviembre de 2011

D81766

ORACLE®

Autor

Jill Moritz

Kenneth Somerville

Cindy Church

Colaboradores y revisores técnicos

Mike Williams

Tom McGinn

Matt Heimer

Joe Darcy

Brian Goetz

Alex Buckley

Adam Messenger

Steve Watts

Redactores

Smita Kommini

Aju Kumar

Richard Wallis

Diseñadores gráficos

Seema M. Bopaiah

Rajiv Chandrabhanu

Editores

Giri Venugopal

Jayanthi Keshavamurthy

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Exención de responsabilidad

Este documento contiene información propiedad de Oracle Corporation y se encuentra protegido por el copyright y otras leyes sobre la propiedad intelectual. Usted sólo podrá realizar copias o imprimir este documento para uso exclusivo por usted en los cursos de formación de Oracle. Este documento no podrá ser modificado ni alterado en modo alguno. Salvo que la legislación del copyright lo considere un uso excusable o legal o "fair use", no podrá utilizar, compartir, descargar, cargar, copiar, imprimir, mostrar, representar, reproducir, publicar, conceder licencias, enviar, transmitir ni distribuir este documento total ni parcialmente sin autorización expresa por parte de Oracle.

La información contenida en este documento puede someterse a modificaciones sin previo aviso. Si detecta cualquier problema en el documento, le agradeceremos que nos lo comunique por escrito a: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 EE. UU. No se garantiza que este documento se encuentre libre de errores.

Aviso sobre restricción de derechos

Si este software o la documentación relacionada se entrega al Gobierno de EE.UU. o a cualquier entidad que adquiera licencias en nombre del Gobierno de EE.UU. se aplicará la siguiente disposición:

U.S. GOVERNMENT RIGHTS

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

Disposición de marca comercial registrada

Oracle y Java son marcas comerciales registradas de Oracle y/o sus filiales. Todos los demás nombres pueden ser marcas comerciales de sus respectivos propietarios.

Contenido

1 Introducción

- Objetivos del curso 1-2
- Programa 1-5
- Instalaciones de su ubicación 1-7
- Prueba 1-8
- Entorno del curso 1-9
- Resumen 1-10

2 Introducción a la tecnología Java

- Objetivos 2-2
- Temas 2-4
- Puesto de Java en el mundo 2-5
- Escritorios de Java 2-6
- Teléfonos móviles de Java 2-7
- Java TV y Java Card 2-8
- Historia de Java 2-9
- Conceptos clave del lenguaje de programación Java 2-10
- Programación de procedimiento 2-11
- Orientado a objetos 2-12
- Distribuido 2-13
- Sencillo 2-14
- Multithread 2-15
- Seguro 2-16
- Programas dependientes de la plataforma 2-17
- Programas independientes de la plataforma 2-20
- Prueba 2-22
- Temas 2-23
- Identificación de grupos de productos de tecnología Java 2-24
- Java SE 2-25
- Java EE 2-26
- Java ME 2-27
- Java Card 2-28
- Configuración del entorno de desarrollo Java 2-29
- Descarga e instalación del JDK 2-30

Examen del Java Development Kit instalado 2-31
 Temas 2-32
 Uso de un entorno de desarrollo integrado 2-33
 Descarga de NetBeans IDE 2-34
 NetBeans IDE y el asistente New Project 2-35
 Prueba 2-36
 Temas 2-37
 Etapas del ciclo de vida del producto 2-38
 Resumen 2-40
 Visión general de la práctica 2-1: Ejecución de un
 programa Java mediante la línea de comandos 2-42
 Visión general de la práctica 2-2: Ejecución de un
 programa Java mediante NetBeans IDE 2-43

3 Consideraciones sobre los objetos

Objetivos 3-2
 Importancia 3-3
 Temas 3-4
 Análisis de un problema mediante el análisis orientado a objetos 3-5
 Proceso de pedido de Duke's Choice 3-6
 Temas 3-7
 Identificación de un dominio de problemas 3-8
 Temas 3-9
 Identificación de objetos 3-10
 Temas 3-13
 Criterios adicionales para reconocer objetos 3-14
 Posibles objetos en el caso práctico de Duke's Choice 3-16
 Temas 3-17
 Identificación de atributos y operaciones de objetos 3-18
 Objeto con otro objeto como atributo 3-19
 Posibles atributos y operaciones para objetos en el caso práctico
 de Duke's Choice 3-20
 Temas 3-21
 Solución del caso práctico: Clases 3-22
 Solución del caso práctico: Atributos 3-23
 Solución del caso práctico: Comportamientos 3-25
 Temas 3-27
 Diseño de clases 3-28
 Clases y objetos resultantes 3-29
 Modelado de clases 3-30

Uso del modelado similar a UML 3-32
 Prueba 3-33
 Resumen 3-35
 Visión general de la práctica 3-1: Análisis de un problema
 mediante el análisis orientado a objetos 3-36
 Visión general de la práctica 3-2: Diseño de una solución
 de programación 3-37

4 Introducción al lenguaje Java

Objetivos 4-2
 Temas 4-3
 Importancia 4-4
 Identificación de los componentes de una clase 4-5
 Estructuración de clases 4-6
 Símbolos utilizados en la definición de un origen Java 4-8
 Unión de todo 4-9
 Prueba 4-11
 Declaraciones y asignaciones de campos 4-12
 Comentarios 4-13
 Temas 4-15
 Métodos 4-16
 Temas 4-18
 Palabras clave 4-19
 Temas 4-20
 Creación y uso de una clase de prueba 4-21
 Método `main` 4-22
 Compilación de un programa 4-23
 Ejecución (prueba) de un programa 4-24
 Compilación y ejecución de un programa mediante un IDE 4-25
 Temas 4-26
 Cómo evitar problemas de sintaxis 4-27
 Temas 4-28
 Trabajar con un depurador de IDE 4-29
 Resumen 4-31
 Visión general de la práctica 4-1: Visualización y adición de
 código en un programa Java existente 4-32
 Visión general de la práctica 4-2: Creación y compilación
 de una clase Java 4-33
 Visión general de la práctica 4-3: Exploración del depurador 4-34

5 Declaración, inicialización y uso de variables

Objetivos 5-2

Importancia 5-3

Temas 5-4

Identificación del uso y la sintaxis de las variables 5-5

Usos de las variables 5-7

Declaración e inicialización de variables 5-8

Temas 5-10

Descripción de tipos de dato primitivos 5-11

Tipos primitivos integrales 5-12

Tipos primitivos de coma flotante 5-14

Tipo primitivo textual 5-15

Tipo primitivo lógico 5-17

Temas 5-18

Asignación de nombres a variables 5-19

Asignación de un valor a una variable 5-21

Declaración e inicialización de varias variables en una línea de código 5-22

Métodos adicionales para declarar variables y asignar valores a variables 5-23

Constantes 5-25

Almacenamiento de primitivos y constantes en memoria 5-26

Prueba 5-27

Temas 5-28

Operadores matemáticos estándar 5-29

Operadores de aumento y disminución (++ y --) 5-31

Prioridad de operadores 5-35

Uso de paréntesis 5-38

Temas 5-39

Uso de ampliación y conversión de tipo 5-40

Ampliación 5-42

Conversión de tipo 5-44

Suposiciones del compilador para tipos de dato integrales y de coma flotante 5-47

Tipos de dato de coma flotante y asignación 5-49

Ejemplo 5-50

Prueba 5-51

Resumen 5-52

Visión general de la práctica 5-1: Declaración de variables de campo
en una clase 5-53Visión general de la práctica 5-2: Uso de operadores y conversión
de tipo para evitar la pérdida de datos 5-54

6 Trabajar con objetos

Objetivos 6-2

Temas 6-3

Trabajar con objetos: Introducción 6-4

Acceso a objetos mediante una referencia 6-5

Clase `Shirt` 6-6

Temas 6-7

Trabajar con variables de referencia de objetos 6-8

Declaración e inicialización: Ejemplo 6-9

Trabajar con referencias de objetos 6-10

Referencias a diferentes objetos 6-13

Referencias a diferentes tipos de objetos 6-14

Referencias y objetos en memoria 6-15

Asignación de una referencia a otra 6-16

Dos referencias, un objeto 6-17

Asignación de una referencia a otra 6-18

Prueba 6-19

Temas 6-20

Clase `String` 6-21

Concatenación de cadenas 6-22

Llamadas al método `String` con valores de retorno primitivos 6-26Llamadas al método `String` con valores de retorno de objeto 6-27

Llamadas a métodos que necesitan argumentos 6-28

Temas 6-29

Documentación de la API de Java 6-30

Documentación de la plataforma Java SE 7 6-31

Plataforma Java SE 7: Resumen del método 6-33

Plataforma Java SE 7: Detalles del método 6-34

Métodos `System.out` 6-35Documentación sobre `System.out.println()` 6-36Uso de los métodos `print()` y `println()` 6-37

Temas 6-38

Clase `StringBuilder` 6-39Ventajas de `StringBuilder` sobre `String` para la concatenación (o adición) 6-40`StringBuilder`: Declaración e instanciación 6-41Adición de `StringBuilder` 6-42

Prueba 6-43

Resumen 6-44

Visión general de la práctica 6-1: Creación y manipulación de objetos Java 6-45

Visión general de la práctica 6-2: Uso de la clase `StringBuilder` 6-46

Visión general de la práctica 6-3: Examen de la especificación de la API de Java 6-47

7 Uso de operadores y construcciones de decisión

Objetivos 7-2

Importancia 7-3

Temas 7-4

Uso de operadores relacionales y condicionales 7-5

Ejemplo de ascensor 7-6

Archivo `ElevatorTest.java` 7-8

Operadores relacionales 7-9

Prueba de la igualdad entre cadenas 7-10

Operadores condicionales comunes 7-11

Operador condicional ternario 7-12

Temas 7-13

Creación de construcciones `if` e `if/else` 7-14

Construcción `if` 7-15

Construcción `if`: Ejemplo 7-16

Construcción `if`: Salida 7-18

Sentencias `if` anidadas 7-19

Construcción `if/else` 7-21

Construcción `if/else`: Ejemplo 7-22

Construcción `if/else` 7-24

Temas 7-25

Encadenamiento de construcciones `if/else` 7-26

Temas 7-28

Uso de la construcción `switch` 7-29

Uso de la construcción `switch`: Ejemplo 7-31

Cuándo utilizar construcciones `switch` 7-33

Prueba 7-34

Resumen 7-36

Visión general de la práctica 7-1: Escritura de una clase que utiliza la sentencia `if/else` 7-37

Visión general de la práctica 7-2: Escritura de una clase que utiliza la sentencia `switch` 7-38

8 Creación y uso de matrices

Objetivos 8-2

Temas 8-3

Introducción a las matrices 8-4

Matrices unidimensionales 8-5

Creación de matrices unidimensionales	8-6
Índices y longitud de matriz	8-7
Temas	8-8
Declaración de una matriz unidimensional	8-9
Instanciación de una matriz unidimensional	8-10
Inicialización de una matriz unidimensional	8-11
Declaración, instanciación e inicialización de matrices unidimensionales	8-12
Acceso a un valor de una matriz	8-13
Almacenamiento de matrices en memoria	8-14
Almacenamiento de matrices de referencias en memoria	8-15
Prueba	8-16
Temas	8-18
Uso de la matriz <code>args</code> en el método <code>main</code>	8-19
Conversión de argumentos <code>String</code> en otros tipos	8-20
Temas	8-21
Descripción de matrices bidimensionales	8-22
Declaración de una matriz bidimensional	8-23
Instanciación de una matriz bidimensional	8-24
Inicialización de una matriz bidimensional	8-25
Temas	8-26
Clase <code>ArrayList</code>	8-27
Nombres de clases y sentencia de importación	8-28
Trabajar con una <code>ArrayList</code>	8-29
Prueba	8-30
Resumen	8-31
Visión general de la práctica 8-1: Creación de una clase con una matriz unidimensional de tipos primitivos	8-32
Visión general de la práctica 8-2: Creación y trabajo con una <code>ArrayList</code>	8-33
Visión general de la práctica 8-3: Uso de argumentos de tiempo de ejecución y análisis de la matriz <code>args</code>	8-34

9 Uso de construcciones de bucle

Objetivos	9-2
Temas	9-3
Bucles	9-4
Comportamiento de repetición	9-5
Creación de bucles <code>while</code>	9-6
Bucle <code>while</code> en Elevator	9-7
Tipos de variables	9-8
Bucle <code>while</code> : Ejemplo 1	9-9

Bucle <code>while</code> : Ejemplo 2	9-10
Bucle <code>while</code> con contador	9-11
Temas	9-12
Bucle <code>for</code>	9-13
Desarrollo de un bucle <code>for</code>	9-14
Temas	9-15
Bucle <code>for</code> anidado	9-16
Bucle <code>while</code> anidado	9-17
Temas	9-18
Bucles y matrices	9-19
Bucle <code>for</code> con matrices	9-20
Definición de valores en una matriz	9-21
Bucle <code>for</code> mejorado con matrices	9-22
Bucle <code>for</code> mejorado con <code>ArrayLists</code>	9-23
Uso de <code>break</code> con bucles	9-24
Uso de <code>continue</code> con bucles	9-25
Temas	9-26
Codificación de un bucle <code>do/while</code>	9-27
Temas	9-29
Comparación de construcciones de bucle	9-30
Prueba	9-31
Resumen	9-33
Visión general de la práctica 9-1: Escritura de una clase que utiliza un bucle <code>for</code>	9-34
Visión general de la práctica 9-2: Escritura de una clase que utiliza un bucle <code>while</code>	9-35
Visión general de la práctica de comprobación 9-3: Conversión de un bucle <code>while</code> en un bucle <code>for</code>	9-36
Visión general de la práctica 9-4: Uso de bucles <code>for</code> para procesar una <code>ArrayList</code>	9-37
Visión general de la práctica 9-5: Escritura de una clase que utiliza un bucle <code>for</code> anidado para procesar una matriz bidimensional	9-38
Visión general de la práctica de comprobación 9-6: Adición de un método de búsqueda a <code>ClassMap</code>	9-39

10 Trabajar con métodos y sobrecarga de métodos

Objetivos	10-2
Temas	10-3
Creación y llamada a métodos	10-4
Forma básica de un método	10-5

Llamada a un método en una clase diferente	10-6
Métodos de llamada y de trabajo	10-7
Transferencia de argumentos y devolución de valores	10-8
Creación de un método con un parámetro	10-9
Creación de un método con un valor de retorno	10-10
Llamada a un método en la misma clase	10-11
Transferencia de argumentos a métodos	10-12
Transferencia por valor	10-13
Ventajas del uso de métodos	10-16
Prueba	10-17
Métodos de llamada: Resumen	10-18
Temas	10-19
Utilidades matemáticas	10-20
Métodos estáticos de <code>Math</code>	10-21
Creación de métodos y variables <code>static</code>	10-22
Variables <code>static</code>	10-24
Métodos estáticos y variables estáticas en la API de Java	10-25
Temas	10-27
Firma de método	10-28
Sobrecarga de métodos	10-29
Uso de la sobrecarga de métodos	10-30
Sobrecarga de métodos y la API de Java	10-32
Prueba	10-33
Resumen	10-34
Visión general de la práctica 10-1: Escritura de un método con argumentos y valores de retorno	10-35
Visión general de la práctica de comprobación 10-2: Escritura de una clase que contenga un método sobrecargado	10-36

11 Uso de encapsulación y constructores

Objetivos	11-2
Temas	11-3
Visión general	11-4
Modificador <code>public</code>	11-5
Riesgos del acceso a un campo <code>private</code>	11-6
Modificador <code>private</code>	11-7
Intento de acceso a un campo <code>private</code>	11-8
Modificador <code>private</code> en los métodos	11-9
Interfaz e implantación	11-10
Métodos <code>get</code> y <code>set</code>	11-11

Uso de los métodos setter y getter	11-12
Método setter con comprobación	11-13
Uso de los métodos setter y getter	11-14
Encapsulación: Resumen	11-15
Temas	11-16
Inicialización de un objeto <code>Shirt</code>	11-17
Constructores	11-18
Creación de constructores	11-19
Inicialización de un objeto <code>Shirt</code> con un constructor	11-21
Varios constructores	11-22
Prueba	11-23
Resumen	11-24
Visión general de la práctica 11-1: Implantación de la encapsulación en una clase	11-25
Visión general de la práctica de comprobación 11-2: Adición de validación a la clase <code>DateThree</code>	11-26
Visión general de la práctica 11-3: Creación de constructores para inicializar objetos	11-27

12 Uso de conceptos orientados a objetos avanzados

Objetivos	12-2
Temas	12-3
Jerarquías de clase	12-4
Temas	12-5
Comportamientos comunes	12-6
Duplicación de código	12-7
Herencia	12-8
Sustitución de métodos de superclase	12-9
Superclase <code>Clothing</code> : 1	12-10
Superclase <code>Clothing</code> : 2	12-11
Superclase <code>Clothing</code> : 3	12-12
Declaración de una subclase	12-13
Declaración de una subclase (palabras clave <code>extends</code> , <code>super</code> y <code>this</code>)	12-14
Declaración de una subclase: 2	12-15
Clases abstractas	12-16
Superclase abstracta <code>Clothing</code> : 1	12-17
Superclase abstracta <code>Clothing</code> : 2	12-18
Relaciones de superclases y subclases	12-19
Otro ejemplo de herencia	12-20
Temas	12-21

Tipos de referencia de superclase	12-22
Acceso a funcionalidades de objeto	12-23
Acceso a métodos de clase desde la superclase	12-24
Conversión del tipo de referencia	12-25
Conversión	12-26
Operador <code>instanceof</code>	12-27
Llamadas a métodos polimórficos	12-28
Prueba	12-29
Temas	12-30
Varias jerarquías	12-31
Interfaces	12-32
Implantación de la interfaz <code>Returnable</code>	12-33
Acceso a los métodos de objeto desde la interfaz	12-34
<code>ArrayList</code>	12-35
Interfaz <code>List</code>	12-36
Temas	12-37
Clase <code>Object</code>	12-38
Llamada al método <code>toString()</code>	12-39
Prueba	12-40
Resumen	12-41
Visión general de la práctica 12-1: Creación y uso de superclases y subclases	12-42
Visión general de la práctica 12-2: Uso de una interfaz Java	12-43

13 Manejo de errores

Objetivos	13-2
Temas	13-3
Informe de excepciones	13-4
Devolución de excepciones	13-6
Tipos de excepciones	13-7
<code>OutOfMemoryError</code>	13-8
Temas	13-9
Pila de métodos	13-10
Pila de llamadas: Ejemplo	13-11
Devolución de objetos <code>Throwable</code>	13-12
Trabajar con excepciones en NetBeans	13-14
Captura de una excepción	13-15
Excepción no resuelta	13-16
Excepción impresa en la consola	13-17
Resumen de los tipos de excepciones	13-18

Prueba 13-19
 Temas 13-21
 Excepciones en la documentación de la API de Java 13-22
 Llamada a un método que devuelve una excepción 13-23
 Trabajar con una excepción comprobada 13-24
 Prácticas recomendadas 13-25
 Prácticas no recomendadas 13-26
 Temas 13-28
 Varias excepciones 13-29
 Captura de IOException 13-30
 Captura de IllegalArgumentException 13-31
 Captura de las excepciones restantes 13-32
 Resumen 13-33
 Visión general de la práctica 13-1: Uso de un bloque try/catch para manejar una excepción 13-34
 Visión general de la práctica 13-2: Captura y devolución de una excepción personalizada 13-35

14 Despliegue y mantenimiento de la aplicación Duke's Choice

Objetivos 14-2
 Temas 14-3
 Paquetes 14-4
 Estructura del directorio de paquetes 14-5
 Paquetes en NetBeans 14-6
 Paquetes en el código fuente 14-7
 Temas 14-8
 DukesChoice.jar 14-9
 Definición de la clase principal de un proyecto 14-10
 Creación del archivo JAR con NetBeans 14-11
 Temas 14-13
 Arquitectura de cliente/servidor de dos niveles 14-14
 Arquitectura de cliente/servidor de tres niveles 14-15
 Temas 14-16
 Aplicación Duke's Choice 14-17
 Clase Clothing 14-18
 Niveles de Duke's Choice 14-20
 Ejecución del archivo JAR desde la línea de comandos 14-21
 Visualización de artículos en la línea de comandos 14-22
 Visualización de artículos en la aplicación web de Duke's Choice 14-23
 Temas 14-25
 Mejora de la aplicación 14-26

Adición de un nuevo artículo para su venta	14-27
Implantación de Returnable	14-29
Implantación de constructor	14-30
Clase Suit: Sustitución de <code>getDisplay()</code>	14-31
Implantación de los métodos getter y setter	14-32
Actualización de aplicaciones con la clase Suit	14-33
Prueba de la clase Suit: Línea de comandos	14-34
Prueba de la clase Suit: Aplicación web	14-35
Adición de la clase Suit a la aplicación web	14-36
Resumen	14-37
Sin prácticas para esta lección	14-38
Resumen del curso	14-39

A Referencia rápida de lenguaje Java

B Consejos para UMLet

Interfaz por defecto de UML	B-2
-----------------------------	-----

C Recursos

Java en Oracle Technology Network (OTN)	C-2
Descargas de Java SE	C-3
Documentación de Java	C-4
Comunidad Java	C-5
Comunidad Java: Enfoque extensivo	C-6
Comunidad Java: Java.net	C-7
Tecnologías Java	C-8
Formación de Java	C-9
Oracle Learning Library	C-10
Java Magazine	C-11

Fundacion Proydesa (fundacion@proydesa.org) has a
non-transferable license to use this Student Guide.

1

Introducción

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Fundacion Proydesa (fundacion@proydesa.org) has a
non-transferable license to use this Student Guide.

Objetivos del curso

Al finalizar este curso, debería estar capacitado para:

- Enumerar y describir varias características clave de la tecnología Java, como que está orientada a objetos, es multithread, distribuida, simple y segura
- Identificar diferentes grupos de tecnología Java
- Describir ejemplos de cómo se utiliza Java en aplicaciones, así como productos de consumo
- Describir las ventajas de utilizar un entorno de desarrollo integrado (IDE)
- Desarrollar clases y describir cómo declarar una clase
- Analizar un problema de negocio para reconocer los objetos y las operaciones que forman los bloques integrantes del diseño de programas Java

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos del curso

- Definir el término *objeto* y su relación con una clase
- Mostrar la sintaxis de programación Java
- Escribir un programa Java simple que se compile y ejecute correctamente
- Declarar e inicializar variables
- Enumerar varios tipos de dato primitivos
- Instanciar un objeto y utilizar de forma eficaz variables de referencia de objetos
- Utilizar operadores, bucles y construcciones de decisión
- Declarar e instanciar matrices y ArrayLists y poder iterar con ellas

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos del curso

- Utilizar la documentación Java para buscar Java Foundation Classes
- Declarar un método con argumentos y valores de retorno
- Utilizar la herencia para declarar y definir una subclase de una superclase existente
- Describir cómo se manejan los errores en un programa Java
- Describir cómo desplegar una aplicación Java simple mediante NetBeans IDE

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Programa

- **Primer día**
 - Lección 1: Introducción
 - Lección 2: Introducción a la tecnología Java
 - Lección 3: Consideraciones sobre los objetos
 - Lección 4: Introducción al lenguaje Java
- **Segundo día**
 - Lección 5: Declaración, inicialización y uso de variables
 - Lección 6: Trabajar con objetos
 - Lección 7: Uso de operadores y construcciones de decisión (solo lección)

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Programa

- Tercer día
 - Lección 7: Uso de operadores y construcciones de decisión (solo prácticas)
 - Lección 8: Creación y uso de matrices
 - Lección 9: Uso de construcciones de bucle
- Cuarto día
 - Lección 10: Trabajar con métodos y sobrecarga de métodos
 - Lección 11: Uso de encapsulación y constructores
 - Lección 12: Introducción a conceptos orientados a objetos avanzados
- Quinto día
 - Lección 13: Manejo de errores
 - Lección 14: Despliegue y mantenimiento de la aplicación Duke's Choice

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Instalaciones de su ubicación

- Inscripción, registro, conexión
- Tarjetas de identificación
- Parking
- Teléfonos
- Internet
- Aseos
- Laboratorios
- Almuerzo
- Cocina/aperitivos
- Horas
- Material (papel, bolígrafos y rotuladores)



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Prueba

- a. ¿Cómo se llama?
- b. ¿Cuál es su trabajo y dónde trabaja?
- c. ¿Cuál es el lugar más interesante que ha visitado?
- d. ¿Por qué le interesa Java?



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Entorno del curso



Computadora del aula

Aplicaciones principales

- JDK 7
- NetBeans 7.0.1

Herramientas adicionales

- Firefox
- Glassfish Server
- UMLet
- Guía del alumno
- Guía de actividades
- Archivos de prácticas
- Documentación de API de Java y especificación del lenguaje Java

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En este curso están preinstalados los siguientes productos para las prácticas de las lecciones:

- **JDK 7:** Java SE Development Kit incluye el compilador Java de línea de comandos (`javac`) y Java Runtime Environment (JRE), que proporciona el comando `java` necesario para ejecutar aplicaciones Java.
- **Firefox:** se utiliza un explorador web para ver la documentación HTML (documentación Java) para las bibliotecas de la plataforma Java SE.
- **NetBeans 7.0.1:** NetBeans IDE es una herramienta de desarrollo de software gratuita y de código abierto para los profesionales que crean aplicaciones de empresa, web, de escritorio y móviles.
- **Glassfish Server:** es un servidor de código abierto que se utiliza para desplegar aplicaciones.
- **Guía del alumno:** la guía tiene todo el material que se trata en clase. Además, incluye tres apéndices que proporcionan información adicional: Referencia rápida de lenguaje Java, Consejos para UMLet (UMLet es una herramienta utilizada para el modelado de UML) y Recursos.
- **Guía de actividades y archivos de prácticas:** se trata de recursos que se utilizan durante las prácticas del curso.
- **Documentación de API de Java y especificación del lenguaje Java:** la documentación de API es la especificación de la interfaz de programación de aplicaciones y la especificación del lenguaje describe usos del lenguaje concretos.

Resumen

En esta lección, ha revisado los objetivos del curso y el programa de clases provisional. Ha conocido a sus compañeros y ha obtenido una visión general del entorno informático que utilizará durante el curso.

Disfrute durante los próximos cinco días de los *Conceptos fundamentales de Java SE 7*.

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

2

Introducción a la tecnología Java

ORACLE®

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Fundacion Proydesa (fundacion@proydesa.org) has a
non-transferable license to use this Student Guide.

Objetivos

Al finalizar esta lección, debería estar capacitado para:

- Describir varias características clave de la tecnología Java
- Describir e identificar características de la programación orientada a objetos
- Tratar la diferencia entre los lenguajes compilados e interpretados
- Describir cómo descargar e instalar la plataforma Java
- Describir cómo ejecutar una aplicación Java mediante la línea de comandos
- Identificar las distintas tecnologías Java

**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos

- Relacionar Java con otros lenguajes
- Tratar los diferentes IDE que soportan el lenguaje Java
- Describir cómo descargar e instalar un IDE
- Describir cada fase del ciclo de vida del producto

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Temas

- Descripción de las características clave de Java y la programación orientada a objetos
- Descripción de la tecnología y del entorno de desarrollo Java
- Trabajar con IDE
- Descripción del ciclo de vida del producto

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Puesto de Java en el mundo



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Para poner las cosas en perspectiva, Java es el único lenguaje de desarrollo más utilizado en el mundo hoy en día, con más de 9 millones de desarrolladores que dicen que han pasado al menos algo de su tiempo desarrollando en Java, según un estudio reciente de Evans Data. Eso entre una población mundial de unos 14 millones de desarrolladores.

Escritorios de Java



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

- 1.100 millones de escritorios ejecutan Java (Nielsen Online, Gartner 2010).
- 930 millones de descargas de JRE al año (agosto de 2009–2010): los usuarios finales utilizan JRE (Java Runtime Environment).
- 9,5 millones de descargas de JDK al año (agosto de 2009–2010): los desarrolladores de Java utilizan JDK (Java Development Kit).

Teléfonos móviles de Java



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Todos los teléfonos que no son smartphones (“teléfonos de funciones”) ejecutan Java.

Java TV y Java Card



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

- El 100% de los reproductores Blu-Ray ejecutan Java.
- 71,2 millones de personas se conectan a la web con dispositivos basados en Java (InStat 2010).
- Se fabrican 1.400 millones de Java Cards cada año (InStat 2010).

Historia de Java

Érase una vez...



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

El lenguaje de programación Java (anteriormente Oak) tiene su origen en 1991 como parte de un proyecto de investigación para desarrollar un lenguaje de programación que salvará la distancia de comunicación entre muchos dispositivos de consumo, como grabadores de vídeo (VCR) y televisores. En concreto, un equipo de desarrolladores de software altamente cualificados de Sun (el Green Team, bajo la dirección de James Gosling) quería crear un lenguaje de programación que permitiera a los dispositivos de consumo con distintos procesadores (CPU) compartir las mismas mejoras de software.

Este concepto inicial se frustró después de varios tratos con compañías de dispositivos de consumo sin éxito. El Green Team se vio obligado a buscar otro mercado para su nuevo lenguaje de programación. Afortunadamente, la World Wide Web era cada vez más popular y el Green Team reconoció que el lenguaje Oak era perfecto para desarrollar componentes multimedia web para mejorar páginas web. Estas pequeñas aplicaciones, llamadas applets, se convirtieron en el uso inicial del lenguaje Oak y los programadores que utilizaban Internet adoptaron lo que se convirtió en el lenguaje de programación Java.

El punto decisivo para Java fue en 1995, cuando Netscape incorporó Java en su explorador.

Sabía que... El personaje de la diapositiva es Duke, la mascota de Java. El Duke original lo creó el artista gráfico del Green Team, Joe Palrang.

Conceptos clave del lenguaje de programación Java

- Orientado a objetos
- Distribuido
- Sencillo
- Multithread
- Seguro
- Independiente de la plataforma

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Los términos mostrados en la diapositiva representan conceptos orientados a objetos. Tratará estos términos en profundidad y esto le ayudará a crear una base para comprender la tecnología Java.

Programación de procedimiento

La programación de procedimiento se basa en la secuencia.

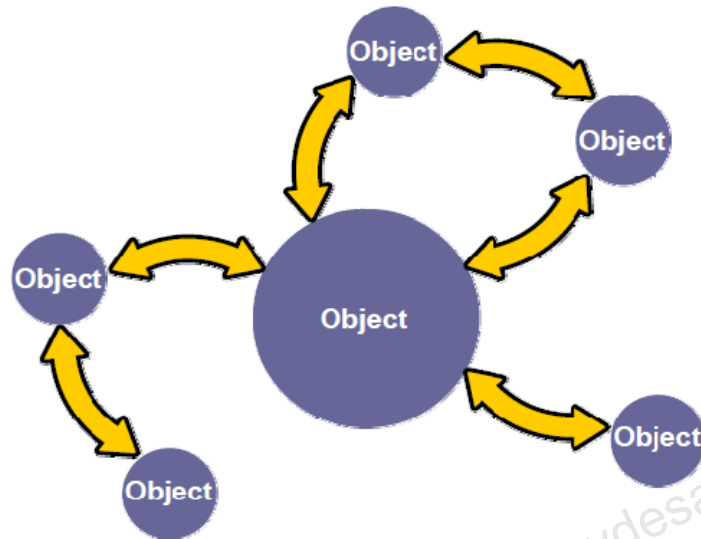
**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

El lenguaje de programación Java es un lenguaje de programación orientada a objetos porque uno de los objetivos principales del programador de tecnología Java es crear objetos (partes de código autónomo) que pueden interactuar con otros objetos para solucionar un problema. La programación orientada a objetos se inició con el lenguaje de programación SIMULA-67 en 1967 y ha llevado a lenguajes de programación populares como, por ejemplo, C++, en el que se basa libremente el lenguaje de programación Java.

En el diagrama se muestra el enfoque en la secuencia del programa de procedimiento.

Orientado a objetos



ORACLE

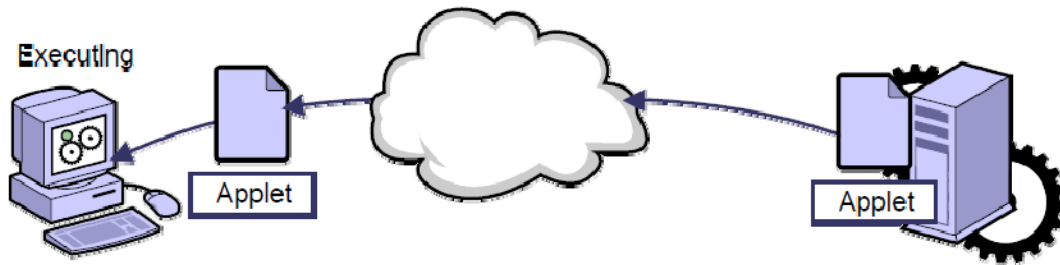
Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

La programación orientada a objetos se diferencia de la programación de procedimiento porque ésta hace hincapié en la secuencia de los pasos de codificación necesarios para solucionar un problema, mientras que la programación orientada a objetos lo hace en la creación e interacción de los objetos.

- **Capacidad de organización en módulos:** el código fuente de un objeto se puede escribir y mantener independientemente del código fuente de otros objetos. Una vez creado, un objeto se puede transferir fácilmente dentro del sistema.
- **Ocultación de información:** al interactuar solo con métodos de un objeto, los detalles de su implantación interna permanecen ocultos al mundo exterior.
- **Reutilización de código:** si un objeto ya existe (quizá escrito por otro desarrollador de software), puede utilizar dicho objeto en su programa. Esto permite a los especialistas implantar, probar y depurar objetos complejos específicos de tareas, en los que puede confiar para ejecutar su propio código.
- **Facilidad de conexión y depuración:** si se descubre que un objeto concreto es problemático, simplemente puede eliminarlo de su aplicación y conectar otro como sustitución. Esto es análogo a la corrección de problemas mecánicos en el mundo real. Si se rompe un tornillo, reemplaza el tornillo, no toda la máquina.

En el diagrama se ilustra el enfoque en objetos e interacciones de objetos del programa orientado a objetos.

Distribuido

**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

El lenguaje de programación Java es un lenguaje distribuido porque proporciona soporte para tecnologías de red distribuidas, como Remote Method Invocation (RMI), Common Object Request Broker Architecture (CORBA) y el localizador uniforme de recursos (URL). Asimismo, las capacidades de carga de clases dinámica de la tecnología Java permiten descargar las partes de código a través de Internet y ejecutarlas en una computadora personal.

Nota: los términos *tecnología Java* y *lenguaje de programación Java* no hacen referencia a lo mismo. La *tecnología Java* hace referencia a una familia de productos de tecnología Java, de los cuales el lenguaje de programación es solo una parte.

Sencillo

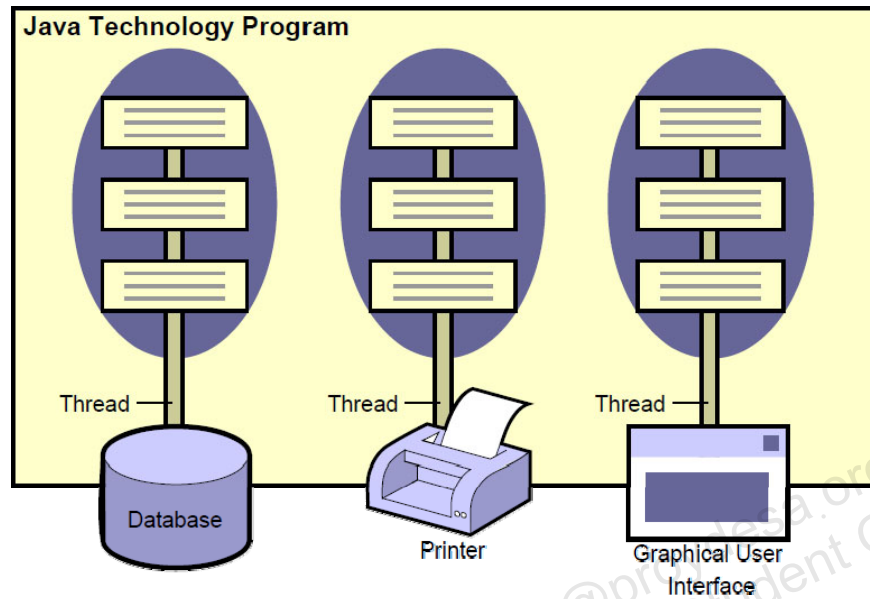
- Se utilizan referencias en lugar de punteros de memoria.
- Un tipo de dato `boolean` puede tener un valor `true` o `false`.
- La gestión de memoria es automática.

The Oracle logo, consisting of the word "ORACLE" in a bold, sans-serif font, is positioned on the right side of a red horizontal bar.

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

El lenguaje de programación Java es sencillo porque los diseñadores han eliminado algunas de las construcciones de programación complejas y poco transparentes encontradas en otros lenguajes de programación populares. Como ejemplo, el lenguaje de programación Java no permite a los programadores manipular directamente punteros a ubicaciones de memoria (una característica compleja de los lenguajes de programación C y C++). En su lugar, el lenguaje de programación Java permite a los programadores manipular solo objetos mediante referencias de objetos. El lenguaje de programación también utiliza una función llamada *recolector de basura* para supervisar y eliminar objetos a los que ya no se hace referencia. Otra característica que hace que el lenguaje de programación Java sea sencillo es que un booleano de Java solo puede tener un valor `true` o `false`, a diferencia de otros lenguajes en los que un booleano se representa con 0 y 1.

Multithread

**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

El lenguaje de programación Java soporta multithread. Esto permite ejecutar varias tareas simultáneamente (al mismo tiempo), como consultar una base de datos, realizar cálculos de larga ejecución y complejos, y mostrar una interfaz de usuario. La capacidad multithread permite a un programa de tecnología Java ser muy eficaz en el uso de recursos del sistema. En la imagen se ilustra cómo es multithread el lenguaje de programación Java.

Seguro



ORACLE

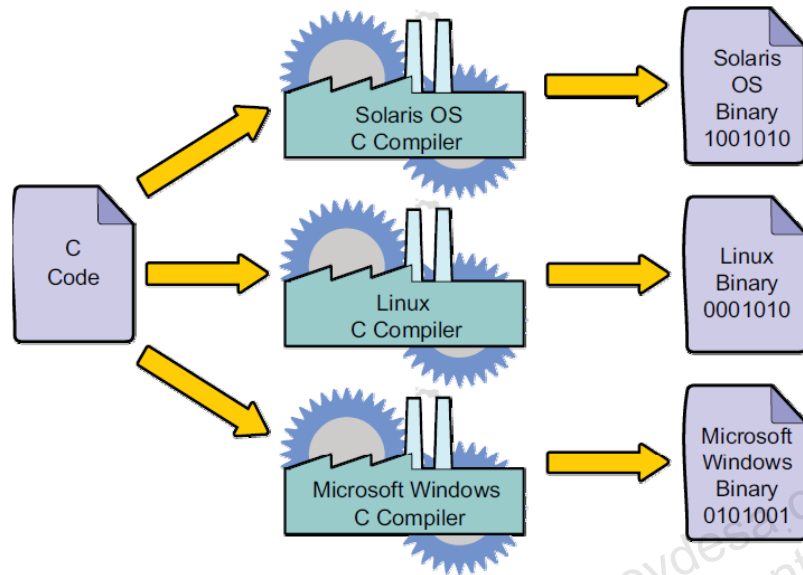
Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Los programas de tecnología Java son seguros porque el lenguaje de programación Java, con el entorno en el que se ejecutan los programas de tecnología Java, utilizan medidas de seguridad para proteger los programas frente a ataques. Estas medidas incluyen:

- Prohibición de que programas distribuidos, como applets, lean y escriban en un disco duro de una computadora.
- Verificación de que todos los programas de tecnología Java contienen código válido.
- Soporte de firmas digitales. El código de tecnología Java lo puede “firmar” una compañía o una persona de forma que otra persona que reciba el código pueda verificar la legitimidad del mismo.
- Prohibición de la manipulación de memoria mediante el uso de punteros.

En la imagen se ilustra cómo se protegen los programas de tecnología Java al no permitir que se ejecute código no válido en una computadora.

Programas dependientes de la plataforma



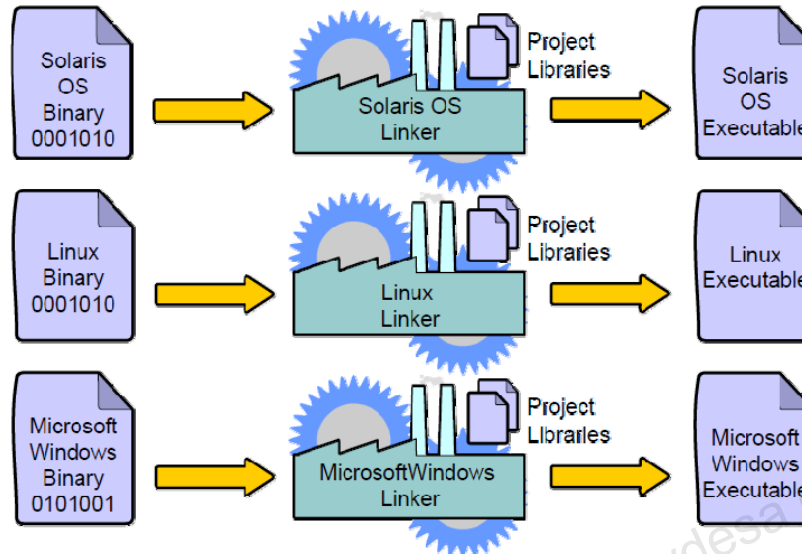
ORACLE®

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Los programas escritos en la mayoría de lenguajes suelen necesitar numerosas modificaciones para ejecutarse en más de un tipo de plataforma de computación (una combinación de CPU y sistema operativo). Esta dependencia de la plataforma se debe a que la mayoría de lenguajes necesitan que escriba código específico de la plataforma subyacente. Los lenguajes de programación populares, como C y C++, necesitan que los programadores compilen y enlacen sus programas, cuyo resultado es un programa ejecutable único en una plataforma. Un compilador es una aplicación que convierte un programa que escribe en un código específico de CPU denominado *código de máquina*. Estos archivos específicos de la plataforma (archivos binarios) a menudo se combinan con otros archivos, como bibliotecas de código escrito previamente, que utilizan un enlace para crear un programa dependiente de la plataforma, denominado *ejecutable*, que puede ejecutar un usuario final. A diferencia de C y C++, el lenguaje de programación Java es independiente de la plataforma.

En la imagen se ilustra cómo un compilador crea un archivo binario.

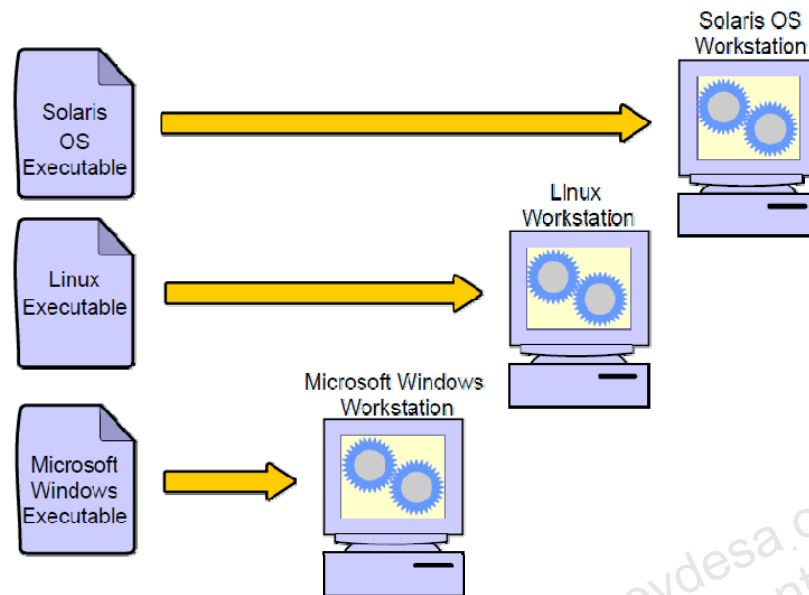
Programas dependientes de la plataforma

**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En la imagen se ilustra cómo se enlaza un archivo binario a bibliotecas para crear un ejecutable dependiente de la plataforma.

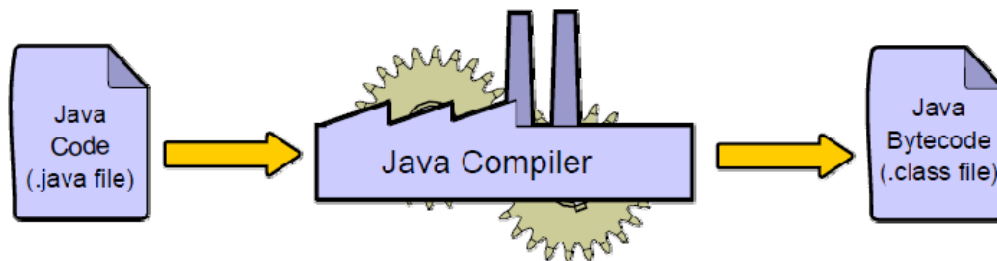
Programas dependientes de la plataforma

**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En la imagen se ilustra cómo los ejecutables dependientes de la plataforma se pueden ejecutar solo en una plataforma.

Programas independientes de la plataforma

**ORACLE**

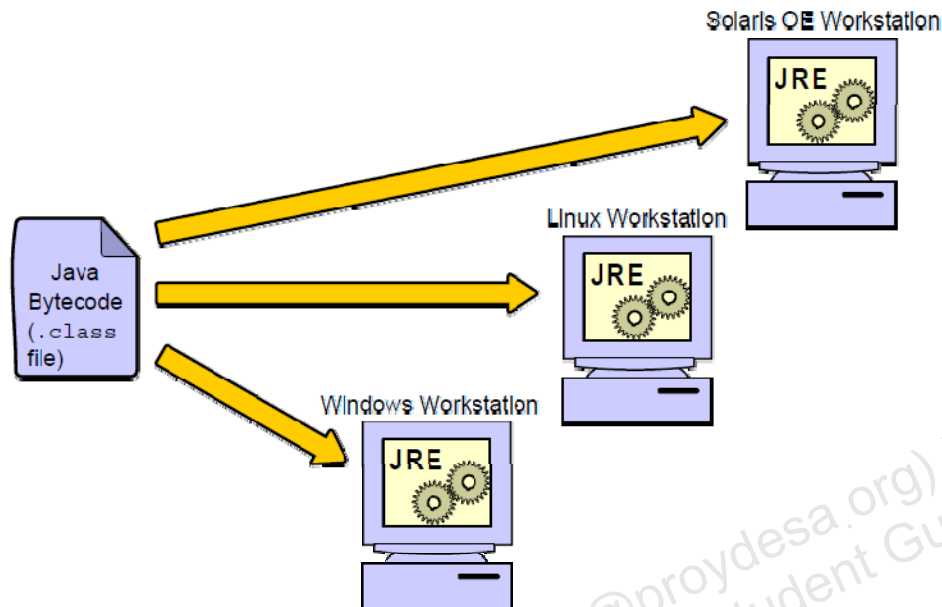
Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En la imagen se ilustra el compilador de tecnología Java (compilador Java) que crea código de byte Java.

Un programa de tecnología Java se puede ejecutar en varias combinaciones de CPU y sistemas operativos distintas, como el sistema operativo Solaris en un chip SPARC, MacOS X en un chip Intel y Microsoft Windows en un chip Intel, normalmente con pocas o ninguna modificación.

Los programas de tecnología Java se compilan mediante un compilador de tecnología Java. El formato resultante de un programa de tecnología Java compilado es código de byte de tecnología Java independiente de la plataforma en lugar de código de máquina específico de CPU. Una vez creado el código de byte, lo interpreta (ejecuta) un intérprete de código de byte denominado *máquina virtual* o VM. Una máquina virtual es un programa específico de la plataforma que comprende el código de byte independiente de la plataforma y puede ejecutarlo en una plataforma concreta. Por este motivo, el lenguaje de programación Java se suele denominar lenguaje interpretado y se dice que los programas de tecnología Java son portátiles o ejecutables en cualquier plataforma. Otros lenguajes interpretados incluyen Perl.

Programas independientes de la plataforma



ORACLE®

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En la imagen se ilustra la ejecución de un archivo de código de byte de tecnología Java (código de byte Java) en varias plataformas donde existe Java Runtime Environment.

Una máquina virtual obtiene su nombre porque es una parte de software que ejecuta código, una tarea normalmente realizada por la CPU o la máquina de hardware. Para que los programas de tecnología Java sean independientes de la plataforma, es necesaria una máquina virtual denominada Java Virtual Machine (JVM) en cada plataforma donde se ejecutará la programación. Java Virtual Machine es responsable de interpretar el código de tecnología Java, cargar las clases Java y ejecutar programas de tecnología Java.

Sin embargo, un programa de tecnología Java necesita que se ejecute más de una Java Virtual Machine.

Un programa de tecnología Java también necesita un juego de bibliotecas de clases Java estándar para la plataforma. Las bibliotecas de clases Java son bibliotecas de código escrito previamente que se puede combinar con el código que escribe para crear aplicaciones sólidas.

Combinados, el software JVM y las bibliotecas de clases Java se denominan Java Runtime Environment (JRE). Java Runtime Environment está disponible en Oracle para muchas plataformas comunes.

Nota: pueden ser necesarias algunas modificaciones para que un programa de tecnología Java sea independiente de la plataforma. Por ejemplo, puede ser necesario modificar los nombres de directorios para que utilicen los delimitadores adecuados (barras inclinadas e invertidas) para el sistema operativo subyacente.

Prueba

Se dice que el lenguaje de programación Java es independiente de la plataforma porque:

- a. El código compilado se ejecuta en varias plataformas con pocas o ninguna modificación.
- b. No permite el uso de punteros para manipular la memoria.
- c. El formato de un programa Java compilado es código específico de CPU.
- d. Es multithread.



Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Respuesta: a

- b es una afirmación correcta pero una respuesta incorrecta porque no está relacionada con la independencia de la plataforma.
- c es incorrecta porque un programa Java compilado no es código específico de CPU. Lo interpreta la máquina virtual que reside en el sistema.
- d es una afirmación correcta porque Java es multithread, pero no es el motivo por el que se dice que es independiente de la plataforma.

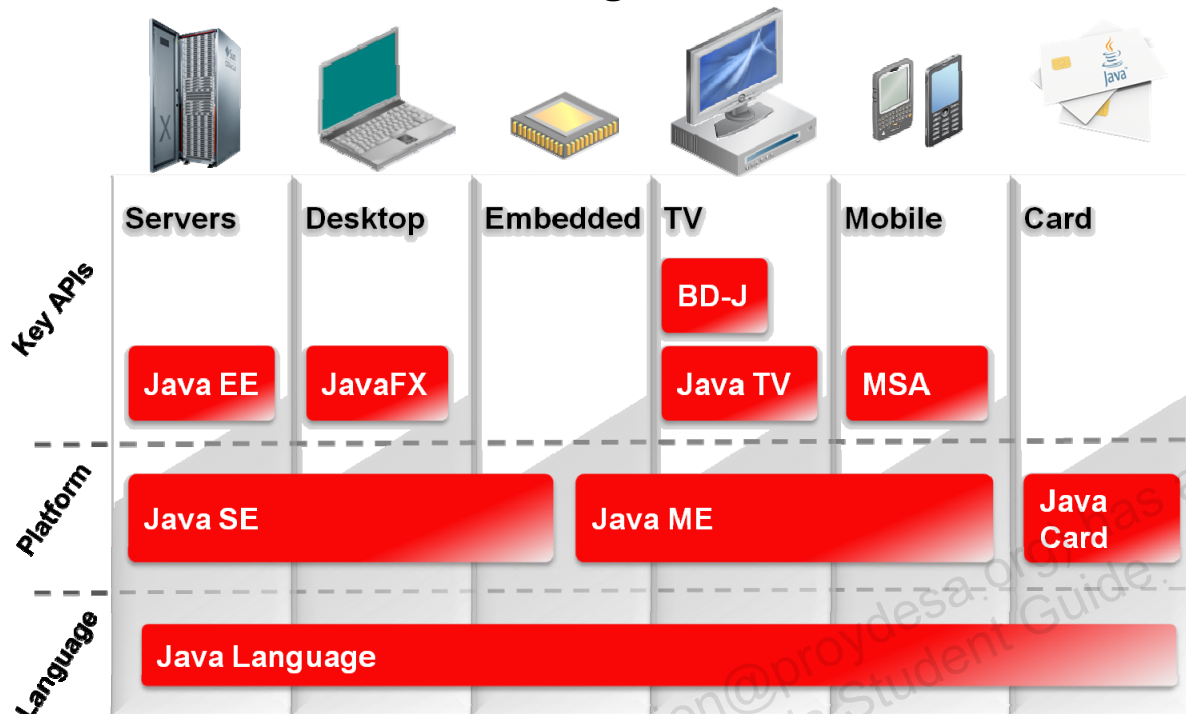
Temas

- Descripción de las características clave de Java y la programación orientada a objetos
- Descripción de la tecnología y del entorno de desarrollo Java
- Trabajar con IDE
- Descripción del ciclo de vida del producto

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Identificación de grupos de productos de tecnología Java



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Oracle proporciona una línea completa de productos de tecnología Java, que van desde kits que crean programas de tecnología Java hasta entornos de emulación (prueba) para dispositivos de consumo como teléfonos móviles. Como se indica en el gráfico, todos los productos de tecnología Java comparten la base del lenguaje Java. Las tecnologías Java, como Java Virtual Machine, se incluyen (de distintas formas) en tres grupos diferentes de productos, cada uno diseñado para cumplir las necesidades de un mercado objetivo concreto. En la figura se ilustran los tres grupos de productos de tecnología Java y sus tipos de dispositivo objetivo. Cada edición incluye un Java Development Kit (JDK) [también denominado Software Development Kit (SDK)] que permite a los programadores crear, compilar y ejecutar programas de tecnología Java en una plataforma concreta.

Nota: la API de JavaFX es un completo cliente para crear interfaces de usuario para el programa Java. La API de MSA es la aplicación de software móvil utilizada para crear interfaces de usuario en dispositivos portátiles.

Java SE

Se utiliza para desarrollar applets que se ejecutan en exploradores web y aplicaciones que se ejecutan en computadoras de escritorio.

**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Java Platform, Standard Edition (Java SE) se utiliza para desarrollar applets y aplicaciones que se ejecutan en exploradores web y en computadoras de escritorio. Por ejemplo, puede utilizar Java SE JDK para crear un programa de procesador de texto para una computadora personal.

Utilizamos dos aplicaciones de escritorio Java en este curso: NetBeans y UMLet.

Nota: los applets y las aplicaciones se diferencian en varios aspectos. Principalmente, los applets se inician en un explorador web, mientras que las aplicaciones se inician en un sistema operativo. Aunque este curso se centra principalmente en el desarrollo de aplicaciones, la mayor parte de la información se puede aplicar al desarrollo de applets.

Java EE

Se utiliza para crear grandes aplicaciones distribuidas de empresa, de servidor y de cliente.

**ORACLE**

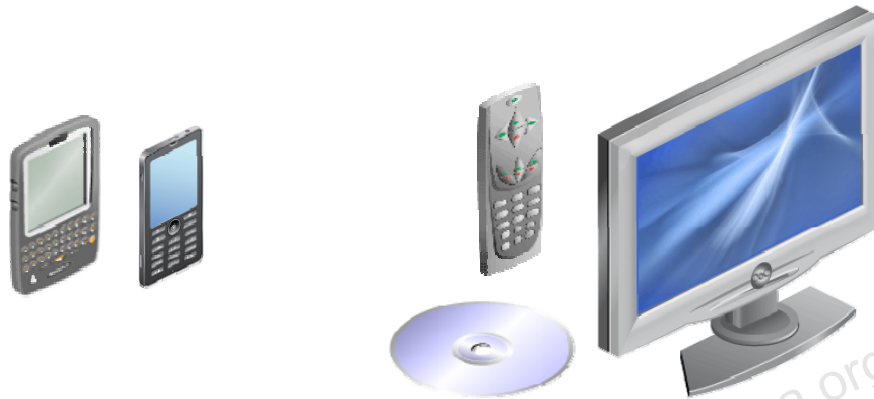
Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Java Platform, Enterprise Edition (Java EE) se utiliza para crear grandes aplicaciones distribuidas de empresa, de servidor y de cliente. Por ejemplo, puede utilizar Java EE JDK para crear una aplicación de compras web (eCommerce) para el sitio web de una compañía minorista.

Java EE se crea sobre la plataforma Java SE, ampliándola con API adicionales que soportan las necesidades de software de empresa de gran escala y de alto rendimiento. Las API se empaquetan y agrupan para soportar distintos tipos de contenedores, como un contenedor web para aplicaciones basadas en web, un contenedor de cliente para clientes gruesos y el contenedor EJB para ejecutar componentes Java eficaces. Algunos tipos de funcionalidades soportados por las distintas API incluyen objetos, IU, integración, persistencia, transacciones y seguridad.

Java ME

Se utiliza para crear aplicaciones para dispositivos de consumo con recursos restringidos.

**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Java Platform, Micro Edition (Java ME) se utiliza para crear aplicaciones para dispositivos de consumo con recursos restringidos. Por ejemplo, puede utilizar Java ME JDK para crear un juego que se ejecute en un teléfono móvil. Las aplicaciones Java de disco Blu-Ray y Java TV utilizan el mismo SDK que Java ME.

Java Card

Java Card se suele utilizar en las siguientes áreas (y muchas más):

- Identidad
- Seguridad
- Transacciones
- SIM de teléfonos móviles



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Configuración del entorno de desarrollo Java

Es fácil configurar el entorno de desarrollo Java.

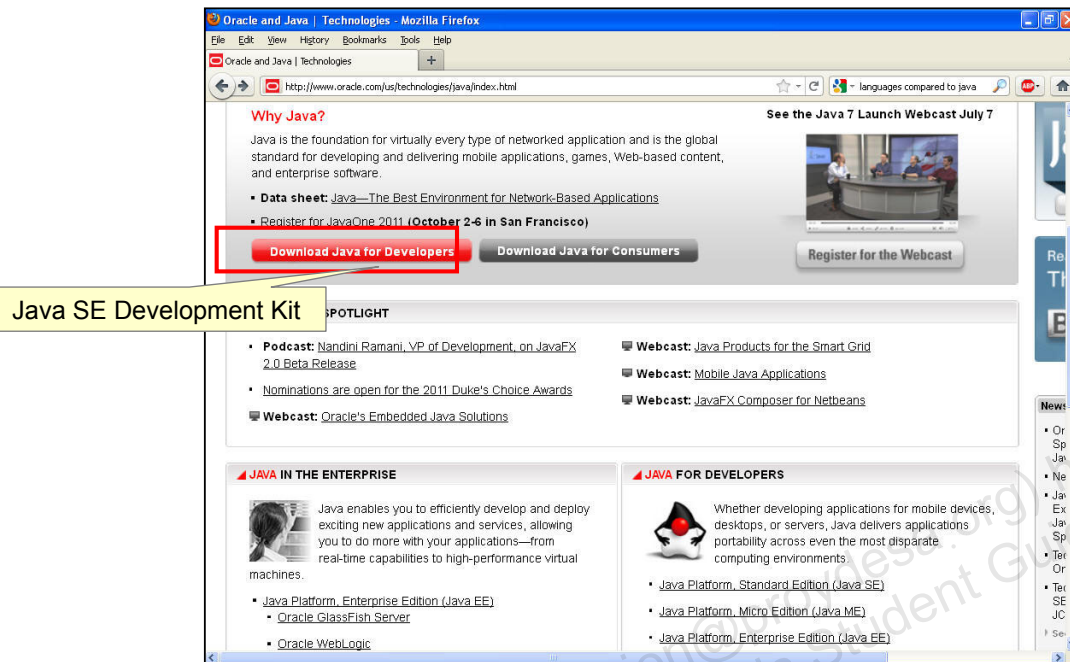
1. Descargue e instale el Java Development Kit (JDK) de oracle.com/java.
2. Defina el valor `PATH` para el JDK instalado.
3. Compile y ejecute una aplicación Java mediante la línea de comandos.

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

La configuración del entorno de desarrollo Java es una tarea sencilla. El JDK está disponible de forma gratuita en el sitio web de Oracle Java.

Descarga e instalación del JDK



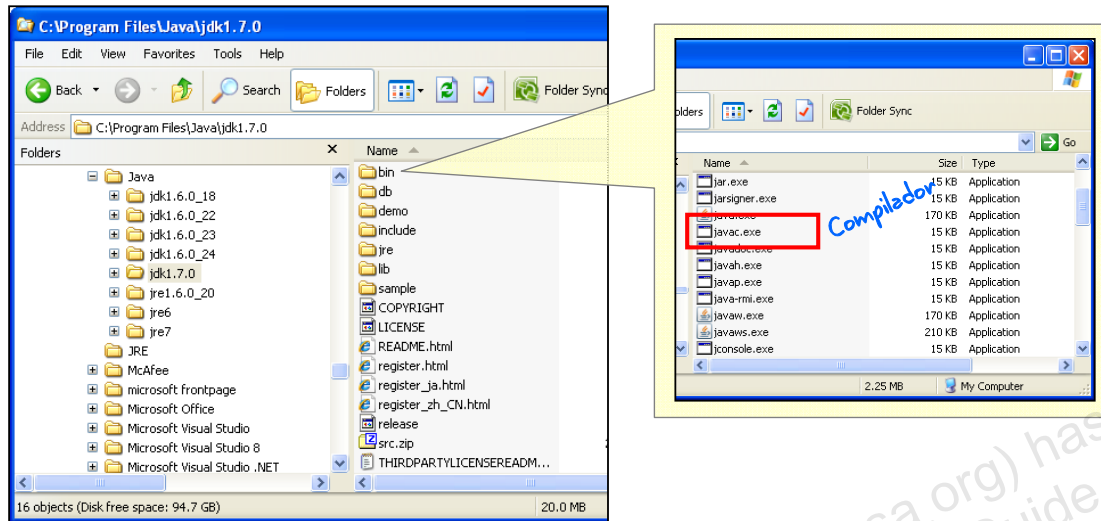
ORACLE®

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

1. Vaya a oracle.com/java.
2. Seleccione el enlace Java Platform, Standard Edition (Java SE).
3. Descargue la versión para su plataforma.
4. Siga las instrucciones de instalación.
5. Defina el valor `PATH` de Java.
6. Compile y ejecute una aplicación Java de ejemplo.

Nota: en las actividades de práctica de esta lección se muestra cómo realizar los pasos 5 y 6.

Examen del Java Development Kit instalado



ORACLE®

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Java SE Development Kit

Al descargar e instalar Java SE Development Kit, se instalan los siguientes elementos:

- Java Runtime Environment (JRE)
- Java Virtual Machine (JVM) para la plataforma que elija
- Bibliotecas de clases Java para la plataforma que elija
- Compilador de tecnología Java
- Utilidades adicionales, como utilidades para crear archivos de almacenamiento Java (archivos JAR) y para depurar programas de tecnología Java
- Ejemplos de programas de tecnología Java

Además de la descarga de Java SDK, hay otros elementos disponibles como descarga independiente, como la documentación de la biblioteca de clases Java (API) y la especificación del lenguaje Java.

Nota: el compilador (`javac`) se encuentra en el directorio `../jdk<version>/bin`.

Temas

- Descripción de las características clave de Java y la programación orientada a objetos
- Descripción de la tecnología y del entorno de desarrollo Java
- **Trabajar con IDE**
- Descripción del ciclo de vida del producto

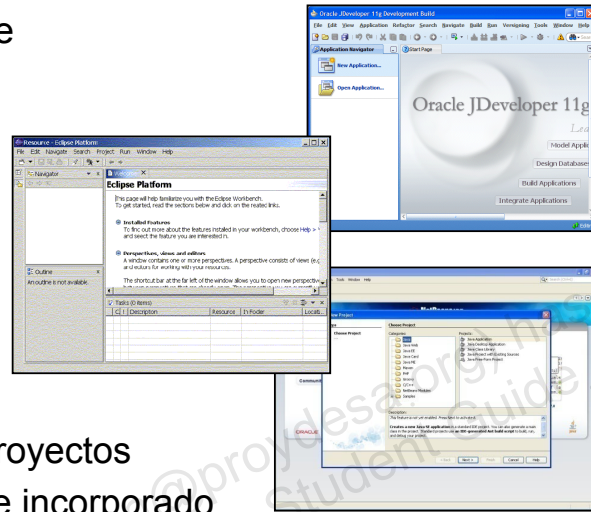
ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de un entorno de desarrollo integrado

Un entorno de desarrollo integrado (IDE) es una herramienta que puede ayudarle con el desarrollo de aplicaciones Java.

- Hay varios IDE disponibles:
 - NetBeans IDE de Oracle
 - JDeveloper de Oracle
 - Eclipse de IBM
- Sus características incluyen:
 - Integración completa
 - Despliegue sencillo
 - Editor inteligente
 - Desarrollo sencillo de proyectos
 - Control de código fuente incorporado



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Los entornos de desarrollo integrados ayudan a reducir el tiempo de desarrollo mediante las siguientes características:

- Automatización de tareas sencillas
- Uso de terminación automática de código
- Integración de la depuración
- Simplificación de la compilación y el despliegue de aplicaciones

Descarga de NetBeans IDE

- Disponible de forma gratuita en NetBeans.org
- Definición automática de las propiedades Java de la aplicación
- Varios grupos disponibles

NetBeans IDE 7.0 Download

6.9.1 | 7.0 | Development | Archive

Email address (optional):

IDE Language: Platform:

Subscribe to newsletters: ☒ Monthly ☐ Weekly

☒ NetBeans can contact me at this address

Note: Greyed out technologies are not supported for this platform.

Supported technologies *

	Java SE	Java EE	C/C++	PHP	All
NetBeans Platform SDK	•	•			•
Java SE	•				•
Java EE		•			•
Java ME					•
Java Card™ 3 Connected					•
C/C++			•		•
Groovy					•
PHP				•	•
Bundled servers					•
GlassFish Server Open Source Edition 3.1		•			•
Apache Tomcat 7.0.11		•			•

Download Download Download Download Download

Free, 66 MB Free, 157 MB Free, 45 MB Free, 42 MB Free, 244 MB

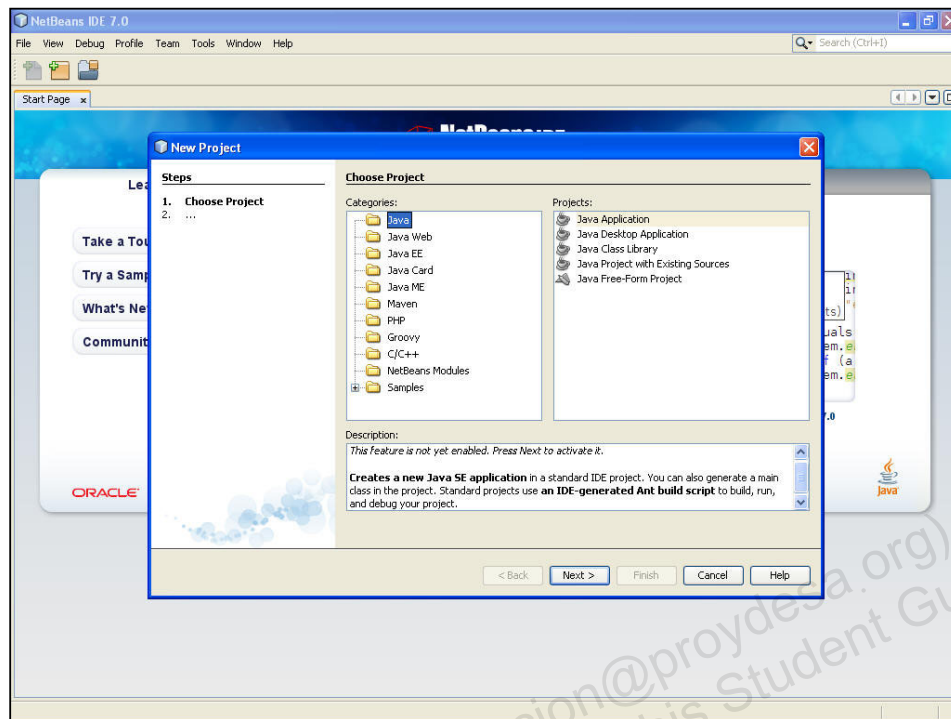
ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Utilizaremos NetBeans IDE para las actividades del curso. Cuando está familiarizado con un IDE, puede aplicar fácilmente sus conocimientos a cualquier IDE similar de su elección.

NetBeans IDE 7.0 (y versiones posteriores) soporta Java SE 7, con el que es compatible. El instalador de IDE necesita un JDK para instalar el IDE en el sistema ya que NetBeans es una aplicación Java. Sin embargo, puede agregar versiones de JDK adicionales después de que NetBeans esté instalado y puede elegir qué versión de JDK utilizar al crear un proyecto de NetBeans. El IDE está disponible con descargas específicas que soportan varias tecnologías Java, como se muestra en el gráfico. Por ejemplo, puede decidir descargar e instalar solo Java SE o puede elegir el grupo **All** de NetBeans.

NetBeans IDE y el asistente New Project



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En NetBeans, trabaja en el contexto de un proyecto, que se compone de un grupo organizado de archivos de origen y metadatos asociados, archivos de propiedades específicos del proyecto, un script Build Ant y una configuración de ejecución, así como todas las herramientas que necesitará para escribir, compilar, probar y depurar la aplicación. Puede crear un proyecto principal con subproyectos así como enlazar proyectos mediante dependencias. Por lo tanto, empezar es tan sencillo como dar un nombre al proyecto. Después de indicar a NetBeans el nombre de un nuevo proyecto, a continuación:

- Crea un árbol de origen con una clase de esqueleto opcional dentro.
- Crea una carpeta para pruebas de unidad.
- Define classpaths para la compilación, ejecución y prueba.
- Define la plataforma Java en la que se ejecuta el proyecto.
- Crea un script Build Ant (`build.xml`), que contiene instrucciones que el IDE utiliza cuando ejecuta comandos en el proyecto, como compilar o ejecutar.

Explorará estas funciones durante las actividades de práctica de esta lección.

Prueba

El grupo de productos de tecnología Java que está diseñado para desarrollar aplicaciones para dispositivos de consumo es

_____.

- a. Java SE JDK
- b. Java ES SDK
- c. Java EE SDK
- d. Java ME SDK

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Respuesta: d

Temas

- Descripción de las características clave de Java y la programación orientada a objetos
- Descripción de la tecnología y del entorno de desarrollo Java
- Trabajar con IDE
- Descripción del ciclo de vida del producto

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Etapas del ciclo de vida del producto

1. Análisis
2. Diseño
3. Desarrollo
4. Prueba
5. Implantación
6. Mantenimiento
7. Fin de vida



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

El ciclo de vida del producto es un proceso iterativo utilizado para desarrollar nuevos productos mediante la solución de problemas.

- **Análisis:** proceso de investigación de un problema que desea solucionar con el producto. Entre otras tareas, el análisis consiste en:
 - Definir claramente el problema que desea solucionar, el nicho de mercado que desea abarcar o el sistema que desea crear. El límite de un problema también se conoce como *ámbito* del proyecto.
 - Identificar los subcomponentes clave del producto general.

Nota: un buen análisis del problema conlleva un buen diseño de la solución y menor tiempo de desarrollo y pruebas.

- **Diseño:** proceso de aplicación de las conclusiones obtenidas durante la etapa de análisis al diseño real del producto. La tarea principal durante la etapa de diseño consiste en desarrollar planos o especificaciones para los productos o componentes del sistema.
- **Desarrollo:** uso de los planos creados durante la etapa de diseño para crear componentes reales.
- **Prueba:** garantía de que los componentes individuales o el producto en su conjunto cumplen los requisitos de la especificación creada durante la etapa de diseño.

Nota: las pruebas las suele realizar un equipo de personas que no son las que han desarrollado realmente el producto. Dicho equipo se asegura de que el producto se prueba sin ninguna parcialidad en nombre del desarrollador.

- **Implantación:** puesta del producto a disposición de los consumidores.
- **Mantenimiento:** solución de problemas con el producto y nueva publicación del producto como una nueva versión o revisión.
- **Fin de vida:** aunque el ciclo de vida del producto no tiene una etapa independiente para el inicio de un concepto o proyecto, sí que la tiene para el fin de un proyecto. El fin de vida consiste en llevar a cabo todas las tareas necesarias para garantizar que los clientes y empleados son conscientes de que un producto ya no se vende o no está soportado, y que hay un nuevo producto disponible.

El ciclo de vida del producto es una parte importante del desarrollo del producto porque ayuda a garantizar que los productos se crean y entregan para que se reduzca el tiempo de comercialización, la calidad del producto es alta y se maximiza el retorno de la inversión. Los desarrolladores que no siguen el ciclo de vida del producto a menudo se encuentran problemas con sus productos que son costosos de arreglar y que se podrían haber evitado.

Resumen

En esta lección, debe haber aprendido lo siguiente:

- Describir varias características clave de la tecnología Java
- Describir e identificar características de la programación orientada a objetos
- Tratar la diferencia entre los lenguajes compilados e interpretados
- Describir cómo descargar e instalar la plataforma Java
- Describir cómo ejecutar una aplicación Java mediante la línea de comandos
- Identificar las distintas tecnologías Java



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Resumen

- Relacionar Java con otros lenguajes
- Tratar los diferentes IDE que soportan el lenguaje Java
- Describir cómo descargar e instalar un IDE
- Describir cada fase del ciclo de vida del producto

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Visión general de la práctica 2-1: Ejecución de un programa Java mediante la línea de comandos

En esta práctica, compilará y ejecutará un programa Java mediante la línea de comandos. Ya se ha creado un programa de tecnología Java. Aprenderá a definir la variable `PATH` para la sesión de `DOS` antes de ejecutar el programa.

**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Visión general de la práctica 2-2: Ejecución de un programa Java mediante NetBeans IDE

En esta práctica, compilará y ejecutará un programa Java mediante el uso de NetBeans IDE. Además, explorará algunas funciones de IDE que permiten desarrollar programas de forma más rápida y sencilla que si utilizara una línea de comandos.

The Oracle logo, consisting of the word "ORACLE" in a bold, sans-serif font, is positioned on the right side of a red horizontal bar.

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Fundacion Proydesa (fundacion@proydesa.org) has a
non-transferable license to use this Student Guide.

Consideraciones sobre los objetos

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos

Al finalizar esta lección, debería estar capacitado para:

- Analizar un problema mediante el análisis orientado a objetos
- Identificar un dominio de problemas
- Identificar los objetos
- Definir criterios adicionales para reconocer objetos
- Definir atributos y operaciones
- Analizar la solución de un caso práctico
- Diseñar una clase
- Modelar una clase



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Importancia

- ¿Cómo decide qué componentes son necesarios para algo que va a crear, como una casa o un mueble?
- ¿Qué es una taxonomía?
- ¿Cómo se relacionan los organismos de una taxonomía?
- ¿Cuál es la diferencia entre atributos y valores?

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Normalmente, primero determina el ámbito del elemento (las dimensiones externas del elemento [altura, ancho, profundidad], la forma en que debe “encajar” el elemento en el entorno [tamaño del lote], etc.). Después de eso, puede empezar a dividir el elemento en sus principales componentes, que se suelen reconocer como sustantivos o “cosas”, como suelo, techo o cocina.

Una taxonomía es una clasificación de organismos relacionados que tienen características (o funciones) similares denominadas atributos, como:

- Aletas o branquias
- Operaciones
- Capacidad de nadar
- Capacidad de caminar sobre dos pies

Los atributos son características o funciones distintivas de un organismo de una taxonomía similar (por ejemplo, una aleta dorsal es un atributo de una ballena).

Los valores representan el estado actual de un atributo. Por ejemplo, una ballena (la ballena azul) tiene una aleta dorsal pequeña mientras que otra (la orca o ballena asesina) tiene una aleta dorsal grande. Grande o pequeña son valores del atributo aleta en la taxonomía de ballena.

Temas

- **Análisis de un problema mediante el análisis orientado a objetos**
- Identificación de un dominio de problemas
- Identificación de los objetos
- Definición de criterios adicionales para reconocer objetos
- Definición de atributos y operaciones
- Análisis de la solución de un caso práctico
- Diseño y modelado de una clase

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Análisis de un problema mediante el análisis orientado a objetos

Duke's Choice vende ropa de su catálogo. El negocio crece un 30% al año y es necesario un nuevo sistema de introducción de pedidos.

**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Duke's Choice produce un catálogo en línea de ropa cada tres meses y lo envía por correo electrónico a sus suscriptores. Cada camisa del catálogo tiene un identificador (ID) de elemento, uno o más colores (cada uno con un código de color), una o más tallas, una descripción y un precio.

Duke's Choice acepta todas las tarjetas de crédito. Los clientes pueden llamar a Duke's Choice para realizar un pedido directamente a un representante del servicio de atención al cliente, o bien pueden rellenar un formulario de pedido en línea en el sitio web de Duke's Choice.

Proceso de pedido de Duke's Choice

**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

A medida que el cliente introduce el pedido en línea, se verifica la disponibilidad de cada elemento (cantidad física). Si uno o más elementos no están disponibles actualmente (en el almacén de Duke's Choice), el elemento se marca como pendiente hasta que llega al almacén. Una vez que todos los elementos están disponibles, se verifica el pago y se envía el pedido al almacén para su ensamblaje y envío a la dirección del cliente. Cuando se recibe el pedido, se da al cliente un ID de pedido, que se utiliza para realizar un seguimiento del pedido a lo largo de todo el proceso. Un representante del servicio de atención al cliente introduce los pedidos que se realizan por teléfono.

Nota: en un análisis real, trabajaría mano a mano con una compañía que obtendría detalles sobre cada aspecto de cómo realiza la compañía su negocio. Este caso práctico resume solo una pequeña parte de la información necesaria para crear un sistema para Duke's Choice.

Temas

- Análisis de un problema mediante el análisis orientado a objetos
- **Identificación de un dominio de problemas**
- Identificación de los objetos
- Definición de criterios adicionales para reconocer objetos
- Definición de atributos y operaciones
- Análisis de la solución de un caso práctico
- Diseño y modelado de una clase

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Identificación de un dominio de problemas

- Un dominio de problemas es el ámbito del problema que va a solucionar.
- Ejemplo: “crear un sistema que permita el método de introducción de pedidos en línea para aceptar y verificar el pago de un pedido”.

The Oracle logo, consisting of the word "ORACLE" in a bold, sans-serif font, is positioned on the right side of a red horizontal bar.

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Puesto que Java es un lenguaje de programación orientada a objetos, uno de los objetivos principales del programador de tecnología Java es crear objetos para crear un sistema o, más concretamente, para solucionar un problema.

El ámbito del problema que solucionará se denomina *dominio de problemas*. La mayoría de los proyectos empiezan con la definición del dominio de problemas, con la recopilación de los requisitos del cliente y la escritura de una sentencia de ámbito que indica brevemente lo que el desarrollador desea lograr. Por ejemplo, una sentencia de ámbito para el proyecto de Duke's Choice puede ser: “crear un sistema que permita el método de introducción de pedidos en línea para aceptar y verificar el pago de un pedido”. Después de determinar el ámbito del proyecto, puede empezar a identificar los objetos que interactuarán para solucionar el problema.

Temas

- Análisis de un problema mediante el análisis orientado a objetos
- Identificación de un dominio de problemas
- **Identificación de los objetos**
- Definición de criterios adicionales para reconocer objetos
- Definición de atributos y operaciones
- Análisis de la solución de un caso práctico
- Diseño y modelado de una clase

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Identificación de objetos

- Los objetos pueden ser físicos o conceptuales.
- Los objetos tienen *atributos* (características) como el tamaño, el nombre, la forma, etc.
- Los objetos tienen *operaciones* (cosas que pueden hacer) como la definición de un valor, la visualización de una pantalla o el aumento de la velocidad.



Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

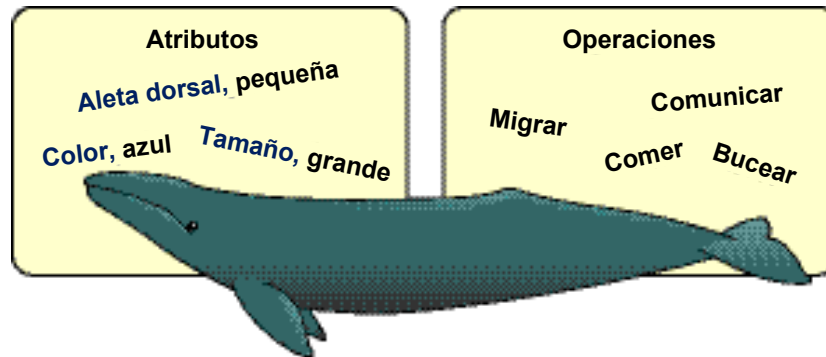
Para validar objetos de un dominio de problemas, primero debe identificar las propiedades de todos los objetos:

- Los objetos pueden ser físicos o conceptuales. Una cuenta de cliente es un ejemplo de un objeto conceptual, porque no es algo que pueda tocar físicamente. Un cajero automático es algo que muchas personas tocan a diario y es un ejemplo de objeto físico.
- Los objetos tienen atributos (características) como el tamaño, el nombre, la forma, etc., que representan el estado del objeto. Por ejemplo, un objeto puede tener un atributo de color. El valor de todos los atributos de un objeto se suele denominar *estado actual* del objeto. Un objeto puede tener un atributo de color con el valor de rojo y un atributo de tamaño con un valor de grande.

- Los objetos tienen operaciones (cosas que pueden hacer) como la definición de un valor, la visualización de una pantalla o el aumento de la velocidad, que representan el comportamiento mediante el cual se puede modificar el estado del objeto. Las operaciones suelen afectar a los atributos de un objeto. Las operaciones que un objeto realiza se suelen denominar *comportamiento*. Por ejemplo, un objeto puede tener una operación que permite a otros objetos cambiar el atributo de color del objeto de un estado a otro, como de rojo a azul.

Sabía que... Los nombres de objetos a menudo son sustantivos, como “cuenta” o “camisa”. Los atributos de objetos a menudo también son sustantivos, como “color” o “tamaño”. Las operaciones de objetos suelen ser verbos o combinaciones de sustantivo y verbo, como “mostrar” o “enviar pedido”. Su capacidad para reconocer objetos en el mundo que le rodea le ayudará a definir mejor los objetos cuando se enfrente a un problema mediante el análisis orientado a objetos.

Identificación de objetos



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En la figura se ilustran las características de una ballena que la convierten en un objeto.

Análisis: mire por la sala. ¿Qué objetos hay en la sala en la que está sentado en este momento? Por ejemplo, una puerta puede ser un objeto del dominio de problemas de “construir una casa”. Una puerta tiene al menos un atributo que tiene un valor (abierto o cerrado) y una operación como “cerrar la puerta” o “abrir la puerta” que permite cambiar el estado de una puerta.

Sabía que... Un atributo con solo dos estados se denomina *atributo booleano*.

Un reloj puede ser un objeto. Un reloj tiene al menos un atributo (hora actual) que tiene un valor (horas:minutos:segundos actuales) y una esfera que permite definir el valor de la hora actual (una operación).

Una silla puede ser un objeto. Una silla tiene al menos un atributo (altura) que tiene un valor (altura en centímetros) y que puede tener una palanca que permita a otro objeto, como una persona, cambiar el valor de altura (una operación). Un instructor puede ser un objeto. Un estudiante puede ser un objeto.

Temas

- Análisis de un problema mediante el análisis orientado a objetos
- Identificación de un dominio de problemas
- Identificación de los objetos
- **Definición de criterios adicionales para reconocer objetos**
- Definición de atributos y operaciones
- Análisis de la solución de un caso práctico
- Diseño y modelado de una clase

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Criterios adicionales para reconocer objetos

- Importancia del dominio de problemas:
 - ¿Existe el objeto en los límites del dominio de problemas?
 - ¿Es necesario el objeto para que se termine la solución?
 - ¿Es necesario el objeto como parte de una interacción entre un usuario y el sistema?
- Existencia independiente

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Utilice los siguientes criterios para seguir probando si algo se debe considerar un objeto de un dominio de problemas:

- Importancia del dominio de problemas
- Existencia independiente

Para determinar si el objeto es relevante para el dominio de problemas, pregúntese lo siguiente:

- ¿Existe el objeto en los límites del dominio de problemas?
- ¿Es necesario el objeto para que se termine la solución?
- ¿Es necesario el objeto como parte de una interacción entre un usuario y la solución?

Nota: algunos elementos de un dominio de problemas pueden ser atributos de objetos o pueden ser los propios objetos. Por ejemplo, la temperatura puede ser un atributo de un objeto de un sistema médico o puede ser un objeto de un sistema científico que realiza un seguimiento de los patrones climatológicos.

Para que un elemento sea un objeto y no un atributo de otro objeto, debe existir independientemente en el contexto del dominio de problemas. Los objetos pueden estar conectados y seguir teniendo una existencia independiente. En el caso práctico de Duke's Choice, un cliente y un pedido están conectados, pero son independientes el uno del otro, por lo que ambos serían objetos.

Al evaluar objetos potenciales, pregúntese si el objeto necesita existir de forma independiente, en lugar de ser un atributo de otro objeto. La identificación de objetos en un dominio de problemas en un arte, no una ciencia. Cualquier objeto puede ser un objeto válido si tiene importancia para el dominio de un problema y tiene las características de un objeto, pero esto no significa que sea un buen objeto. La persona que modela el sistema o la solución debe entender el sistema completo.

Posibles objetos en el caso práctico de Duke's Choice

**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En la figura se ilustran tres objetos del dominio de problemas para el sistema de introducción de pedidos de Duke's Choice. Esta lista no es una respuesta exhaustiva ni acreditada. Esta lista es solo un primer análisis del sistema.

Algunos sustantivos que probablemente no son objetos adecuados para este sistema son:

- Fax
- Verificación
- Pago

Temas

- Análisis de un problema mediante el análisis orientado a objetos
- Identificación de un dominio de problemas
- Identificación de los objetos
- Definición de criterios adicionales para reconocer objetos
- **Definición de atributos y operaciones**
- Análisis de la solución de un caso práctico
- Diseño y modelado de una clase

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Identificación de atributos y operaciones de objetos

- Los atributos son datos, como:
 - ID
 - Objeto de pedido
- Las operaciones son acciones, como:
 - Suprimir elemento
 - Cambiar ID

ORACLE

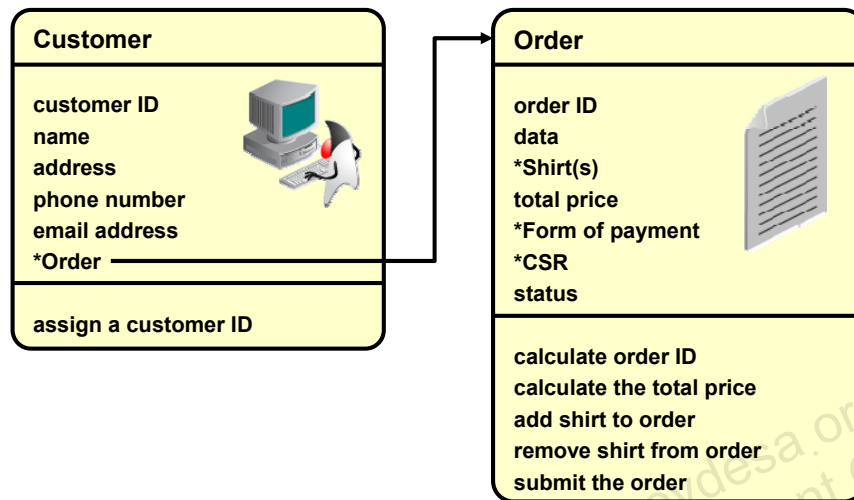
Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Después de identificar los objetos, especifique sus atributos y operaciones.

Como se ha descrito anteriormente, los atributos definen el estado de un objeto. Los atributos pueden ser datos, como un ID de pedido e ID de cliente para un objeto de pedido, o bien pueden ser otro objeto, como el cliente que tiene un objeto de pedido completo como atributo en lugar de solo el ID de pedido.

Como se ha descrito anteriormente, las operaciones son comportamientos que normalmente modifican el estado de un atributo. Por ejemplo, un pedido se puede imprimir, ofrece la posibilidad de agregar o suprimir elementos, etc. (El cliente o el representante del servicio de atención al cliente inicializará esas acciones en la vida real, pero las operaciones pertenecen al objeto de pedido.)

Objeto con otro objeto como atributo



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Un atributo puede ser una referencia a otro objeto. Por ejemplo, el objeto de cliente puede tener un atributo que sea un objeto de pedido. Esta asociación puede ser necesaria o no, según el problema que intenta solucionar.

Nota: utilice nombres de atributos y operaciones que describan claramente el atributo o la operación. En la figura se ilustra el objeto de cliente que contiene un atributo de pedido. Los asteriscos (*) indican atributos que son otros objetos.

Posibles atributos y operaciones para objetos en el caso práctico de Duke's Choice

order ID Order data *Shirt(s) total price *Form of payment *CSR status	shirt ID price description size color code	customer ID Customer name address phone number email address *Order
calculate order ID calculate the total price add shirt to order remove shirt from order submit the order	calculate shirt ID display shirt information	assign a customer ID

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En la figura se ilustran algunos posibles atributos y operaciones para los objetos de pedido, camisa y cliente.

Temas

- Análisis de un problema mediante el análisis orientado a objetos
- Identificación de un dominio de problemas
- Identificación de los objetos
- Definición de criterios adicionales para reconocer objetos
- Definición de atributos y operaciones
- **Análisis de la solución de un caso práctico**
- Diseño y modelado de una clase

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Solución del caso práctico: *Clases*

Clase	Order	Shirt	Customer	Form of Payment	Catalog	CSR
-------	-------	-------	----------	-----------------	---------	-----

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

La tabla de la diapositiva muestra las clases:

- Order
- Shirt
- Customer
- Form of Payment
- Catalog
- CSR

Solución del caso práctico: *Atributos*

Clase	Order	Shirt	Customer
Atributos	order ID date *Shirt(s) total price *Form of payment *CSR status	shirt ID price description size color code	customer ID name address phone number email address *Order

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Solución del caso práctico: *Atributos*

Clase	Form of Payment	Catalog	CSR
<i>Atributos</i>	customer ID name address phone number email address *Order	*Shirt(s)	name extension

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Solución del caso práctico: *Comportamientos*

Clase	Order	Shirt	Customer
Atributos	customer ID name address phone number email address *Order	*Shirt(s)	name extension
Comportamientos	verify credit card number verify check payment	add a shirt remove a shirt	process order

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Solución del caso práctico: *Comportamientos*

Clase	Form of Payment	Catalog	CSR
Atributos	customer ID name address phone number email address *Order	*Shirt(s)	name extension
Comportamientos	verify credit card number verify check payment	add a shirt remove a shirt	process order

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

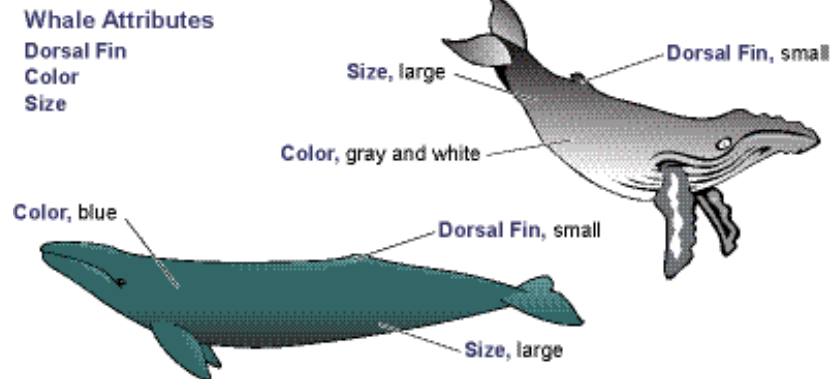
Temas

- Análisis de un problema mediante el análisis orientado a objetos
- Identificación de un dominio de problemas
- Identificación de los objetos
- Definición de criterios adicionales para reconocer objetos
- Definición de atributos y operaciones
- Análisis de la solución de un caso práctico
- **Diseño y modelado de una clase**

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Diseño de clases



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

La identificación de objetos ayuda a diseñar la clase o el plano para cada uno de los objetos de un sistema. Por ejemplo, los fabricantes de ventanas a menudo crean un único plano para cada uno de los estilos de ventanas que crean. Estos planos definen el rango de colores y estilos que se pueden seleccionar cuando se compra la ventana.

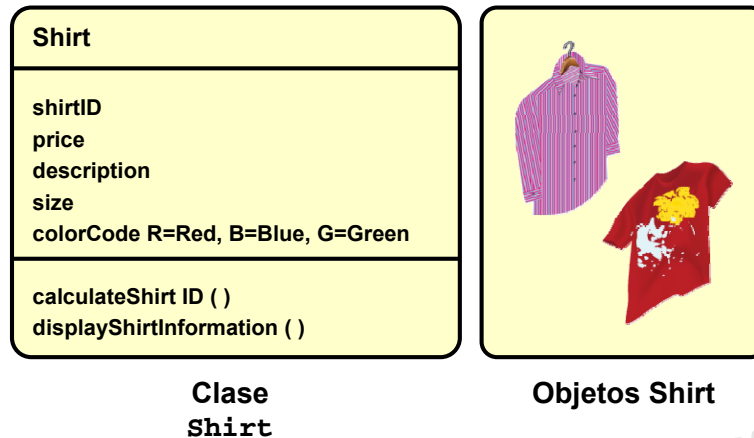
A continuación, estos planos son la base de cualquier número de ventanas con cualquier número de combinaciones de color y estilo. En términos de diseño orientado a objetos, cada objeto (ventana) creado con la clase (plano genérico) se denomina *instancia* de una clase. En concreto, cada objeto creado de una clase puede tener un estado determinado (valores) para cada uno de sus atributos, pero tendrá los mismos atributos y operaciones.

Nota: el diccionario *American Heritage Dictionary* define la palabra *clase* como “un grupo cuyos miembros tienen determinados atributos en común”.

Las clases y los objetos a menudo se utilizan en el campo de la biología. Por ejemplo, a un biólogo marino que estudia criaturas marinas a menudo se le pide que las clasifique en una familia, o clase, de criaturas marinas.

En términos de análisis orientado a objetos, cada animal (como una ballena azul) de una familia (como ballenas) se puede considerar una instancia de objeto de la clase ballena.

Clases y objetos resultantes



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En referencia al caso práctico de Duke's Choice:

- Una clase es la forma de definir un objeto. Las clases son categorías, plantillas o planos descriptivos. `Shirt` puede ser una clase que define todas las camisas que tienen un ID de camisa, tamaño, código de color, descripción y precio.
- Los objetos son instancias únicas de clases. El polo azul grande que cuesta 29,99 \$ con el ID de camisa 62467-B es una instancia de la clase `Shirt`, como lo es la camisa verde pequeña con el mismo precio y el ID de camisa 66889-C, o la camisa de cuadros de 39,99 \$ con el ID 09988-A. También puede tener dos objetos `Shirt` en memoria con exactamente los mismos valores de atributos.

En el gráfico se ilustra una clase y varios objetos basados en la clase.

Nota: volverá a ver la clase `Shirt` a lo largo de este curso.

En el lenguaje de programación Java, los atributos se representan mediante variables y las operaciones se representan mediante métodos. Las variables son el mecanismo del lenguaje de programación Java para contener datos. Los métodos son el mecanismo del lenguaje de programación Java para realizar una operación.

Modelado de clases

Sintaxis:

ClassName
<code>attributeVariableName [range of values]</code> <code>attributeVariableName [range of values]</code> <code>attributeVariableName [range of values]</code> <code>...</code>
<code>methodName()</code> <code>methodName()</code> <code>methodName()</code> <code>...</code>

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

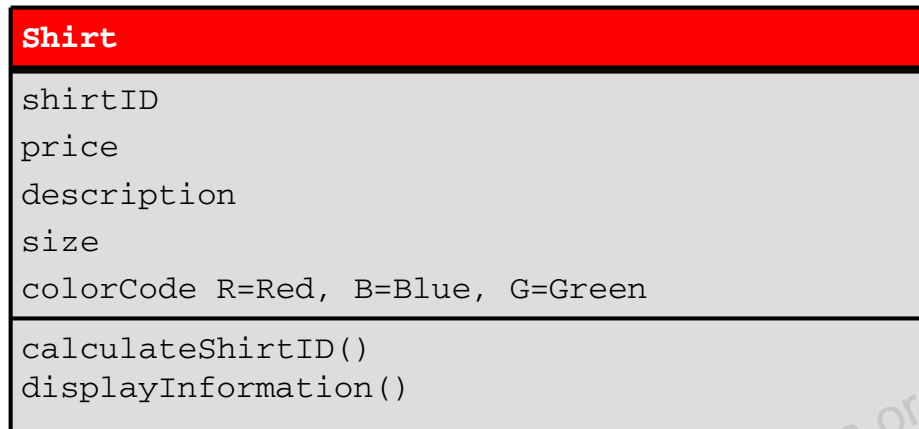
La primera fase de la etapa de diseño consta de la organización visual o el modelado de un programa y sus clases. Cada clase de un diseño se debe modelar para que esté en un cuadro con el nombre de la clase en la parte superior, seguido de una lista de variables de atributos (incluido el rango de los posibles valores) y una lista de métodos.

La sintaxis para modelar una clase se muestra en la figura. La sintaxis utiliza lo siguiente:

- `ClassName` es el nombre de la clase.
- `attributeVariableName` es el nombre de la variable de un atributo.
- `range of values` es un rango opcional de valores que puede contener el atributo.
- `methodName` es el nombre de un método.

Modelado de clases

Ejemplo:



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

La figura contiene un objeto Shirt modelado.

Nota: esta técnica de modelado se basa libremente en una versión ligera de Unified Modeling Language (UML), que es una herramienta para ayudar en el proceso de modelado (algunos de los detalles se han eliminado para los nuevos programadores).

Los nombres de variables y de métodos se escriben en un estilo tipográfico especial denominado “CamelCase”. CamelCase especifica que una variable o método, que representa cualquier atributo u operación de varias palabras, empieza con una letra minúscula y posteriormente las palabras aparecen en mayúscula. Por ejemplo, una operación como “calcular el precio total” se escribe `calcTotalPrice()`. Asimismo, un juego de paréntesis cerrados indica un método.

Nota: el modelado de clases es similar al modelado de estructuras de base de datos. De hecho, los datos de objetos se pueden almacenar en una base de datos mediante la API de Java Database Connectivity (JDBC). La API de JDBC permite leer y escribir registros mediante sentencias de lenguaje de consulta estructurado (SQL) en los programas de tecnología Java.

Uso del modelado similar a UML

UML: Unified Modeling Language

- UML se utiliza para:
 - Modelar los objetos, los atributos, las operaciones y las relaciones en programas orientados a objetos.
 - Modelar el comportamiento dinámico del sistema mostrando colaboraciones entre objetos y cambios en los estados internos de objetos.
- Hay muchos cursos disponibles que enseñan UML.

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

UML se utiliza para modelar programas orientados a objetos. No enseñaremos UML en nuestros cursos de Java, pero podemos mostrar algunas funciones de UML que puede utilizar para solucionar el caso práctico.

- Elija sustantivos para todos los objetos.
- Elija verbos para todos los métodos.
- Elija adjetivos para todos los atributos.

Puede utilizar un editor de texto simple para realizar la práctica 3. Queremos que se acostumbre a buscar los objetos que conforman las clases. UML es un buen método para identificar las clases, los objetos y los métodos que incluye el caso práctico.

Prueba

Elija la respuesta que representa dos propiedades diferentes de un objeto:

- a. Métodos y operaciones
- b. Dominio de problemas
- c. Atributos y operaciones
- d. Variables y datos

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Respuesta: c

Prueba

¿Cuál de las siguientes afirmaciones es cierta?

- a. Un objeto es un plano de una clase.
- b. Un objeto y una clase son exactamente lo mismo.
- c. Un objeto es una instancia de una clase.
- d. Un atributo no puede ser una referencia a otro objeto.

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Respuesta: c

- a es falsa porque una clase es un plano de un objeto.
- b es falsa porque un objeto es simplemente una instanciación de una clase y una clase sirve como plano para el objeto.
- c es correcta.
- d es falsa porque un atributo puede ser una referencia a otro objeto.

Resumen

En esta lección, debe haber aprendido lo siguiente:

- Analizar un problema mediante el análisis orientado a objetos
- Identificar un dominio de problemas
- Identificar los objetos
- Definir criterios adicionales para reconocer objetos
- Definir atributos y operaciones
- Análizar la solución de un caso práctico
- Diseñar una clase
- Modelar una clase



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Visión general de la práctica 3-1: Análisis de un problema mediante el análisis orientado a objetos

En esta práctica, utilizará el análisis orientado a objetos para mostrar los objetos, los atributos y las operaciones de un caso práctico.



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Visión general de la práctica 3-2: Diseño de una solución de programación

En esta práctica, utilizará la tarea de la práctica 3-1 y producirá un diseño mediante una notación similar a UML.

Para crear la notación similar a UML, puede utilizar un editor de texto simple o la herramienta UMLet. En el apéndice B de la *Guía del alumno*, titulado “Consejos para UMLet”, se proporciona una breve visión general de la herramienta UMLet.

The Oracle logo, consisting of the word "ORACLE" in a bold, sans-serif font, is positioned on the right side of a red horizontal bar.

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Fundacion Proydesa (fundacion@proydesa.org) has a
non-transferable license to use this Student Guide.

4

Introducción al lenguaje Java

ORACLE®

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Fundacion Proydesa (fundacion@proydesa.org) has a
non-transferable license to use this Student Guide.

Objetivos

Al finalizar esta lección, debería estar capacitado para:

- Definir una clase
- Identificar los componentes de una clase
- Explicar el término *objeto*
- Describir el objetivo de una variable
- Analizar métodos y describir cómo utilizar un método `main`
- Describir los elementos que componen una clase Java, como las declaraciones, los valores de retorno y el uso correcto de los corchetes y las llaves
- Identificar palabras clave y describir su objetivo
- Probar y ejecutar un programa simple
- Describir algunas causas comunes de errores de sintaxis
- Describir el objetivo y las funciones de un depurador de IDE



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Temas

- Definición de una clase, identificación de componentes de una clase y uso de variables
- Análisis de métodos y el uso de un método `main`
- Identificación de palabras clave
- Prueba y ejecución de un programa Java simple
- Descripción de algunas causas comunes de errores de sintaxis
- Descripción del objetivo y las funciones de un depurador de IDE

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Importancia

¿Cómo prueba algo que ha creado, como una casa, un mueble o un programa?

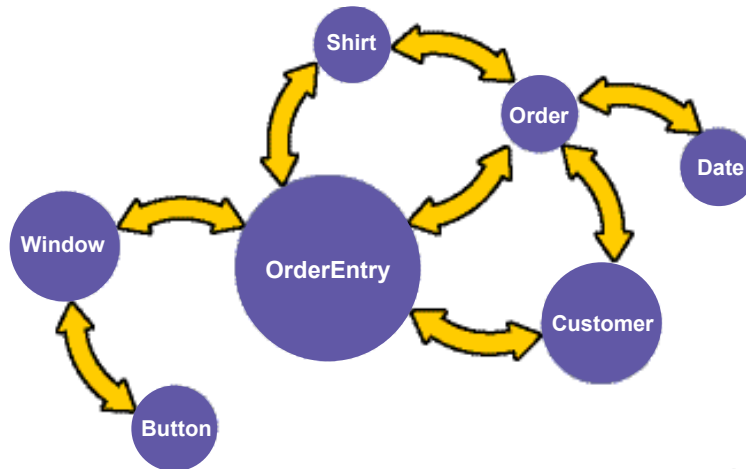
ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En esta lección se proporciona una visión general de los componentes de una clase. También describe cómo compilar y ejecutar un programa de tecnología Java que se compone de varias clases. Necesitamos comprender de qué trata el desarrollo y la prueba de clases.

Hay varias formas de probar un programa. Puede probar distintos componentes (prueba de unidad), puede probar todo el elemento y ver si “encaja” en el entorno, etc.

Identificación de los componentes de una clase



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Las clases son los planos que crea para definir los objetos de un programa. Por ejemplo, en la figura se ilustran algunos de los objetos que pueden existir en el programa de introducción de pedidos para Duke's Choice.

Una aplicación de escritorio suele estar compuesta por un objeto, a menudo denominado objeto controlador, objeto principal u objeto de prueba que es el punto de inicio del programa. En la figura anterior, el objeto OrderEntry puede interactuar con uno o más objetos Window, objetos Customer, objetos Order, etc. mientras se ejecuta el programa. Cada objeto de esta ilustración es una instancia de un plano o una clase. Por ejemplo, todos los objetos Window son instancias de las clases `Window`. Algunas clases, como la clase `Window` (utilizada para crear ventanas de interfaz gráfica de usuario [GUI]), son clases de uso general y se proporcionan como parte de la API de tecnología Java. Otras clases, como la clase `Shirt`, pueden ser únicas para el programa concreto, por lo que debe crearlas. En este curso se describe cómo utilizar las clases existentes y cómo crear y utilizar las propias.

Estructuración de clases

- Declaración de clase
- Declaraciones de campo (los atributos de clase se denominan “campos”)
 - Los campos también se pueden inicializar en el momento de la declaración.
- Métodos (opcionales)
- Comentarios (opcionales)



Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Las clases están formadas por el código de tecnología Java necesario para instanciar objetos, como objetos Shirt. En este curso se divide el código de un archivo de clase Java en cuatro secciones independientes:

- Declaración de clase.
- Declaraciones de campo (los atributos de clase se denominan “campos”). Las variables contienen valores y los valores pueden cambiar durante el transcurso de la aplicación. Los campos son un tipo de variable y las variables locales son otro tipo de variable. Las variables también se pueden inicializar en el momento de la declaración.
- Métodos (opcionales).
- Comentarios (opcionales).

Sabía que... Una clase no tiene que contener métodos y atributos.

Estructuración de clases

```

public class Shirt {
    public int shirtID = 0; // Default ID for the shirt
    public String description = "-description required-"; // default
    // The color codes are R=Red, B=Blue, G=Green, U=Unset
    public char colorCode = 'U';
    public double price = 0.0; // Default price for all shirts
    public int quantityInStock = 0;

    // This method displays the values for an item
    public void displayInformation() {
        System.out.println("Shirt ID: " + shirtID);
        System.out.println("Shirt description:" + description);
        System.out.println("Color Code: " + colorCode);
        System.out.println("Shirt price: " + price);
        System.out.println("Quantity in stock: " + quantityInStock);
    }

    // end of display method
} // end of class

```

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

El código de programación para una clase se incluye en un archivo de texto que debe cumplir una determinada estructura. En el ejemplo se muestra una clase `Shirt` para todas las camisas que aparecerán en el catálogo de Duke's Choice. La clase `Shirt` tiene varios campos y un método, `displayInformation`, para imprimir los valores de los campos.

Símbolos utilizados en la definición de un origen Java

- Llaves **{ }**
- Paréntesis **()**
- Puntos y comas **;**
- Comas **,**
- Comillas simples **' '**
- Comillas dobles **" "**
- Comentario de una línea **//**

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

- Las llaves { } significan un bloque de código. Las llaves incluyen el código de un método concreto o de una clase completa. (Estos son solo algunos de los usos de las llaves).
- Los paréntesis () se utilizan para indicar datos de entrada (también denominados "argumentos") que se pueden transferir a un método.
- Los puntos y comas (;) significan el final de una sentencia.
- Las comas (,) pueden separar varios argumentos y valores.
- Las comillas simples (' ') definen caracteres únicos.
- Las comillas dobles (" ") definen una cadena de varios caracteres.
- Las barras inclinadas dobles (//) indican un comentario de una sola línea.

Unión de todo

- Sintaxis para declarar una clase:

```
[modifiers] class class_identifier
```

- Ejemplo de clase:

```
public class Shirt{
    public double price;

    public void setPrice(double priceArg){
        price = priceArg;
    }
}
```

Diagrama de anotación: Se muestran las llaves de apertura y cierre para la clase `Shirt`. Una línea azul conecta la llave de apertura `{` con el texto "Llaves de apertura y de cierre para la clase Shirt". Otra línea azul conecta la llave de cierre `}` con el mismo texto.

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Debe declarar una clase para cada clase diseñada para el dominio de problemas. Para cada clase, debe escribir una declaración de clase. La sintaxis para declarar una clase es:

```
[modifiers] class class_identifier
```

- La variable `[modifiers]` determina la accesibilidad que otras clases tienen a esta clase. Los modificadores se abordan con más detalle más adelante en este curso. La variable `[modifiers]` es opcional (se indica con corchetes) y puede ser pública, abstracta o final. Por ahora, utilice el modificador `public`.
- La palabra clave `class` indica al compilador que el bloque de código es una declaración de clase. Las palabras clave son palabras reservadas en el lenguaje de programación Java para determinadas construcciones.
- `class identifier` es el nombre que se da a la clase. Las instrucciones de nomenclatura de clases son las siguientes:
 - Los nombres de clases deben ser sustantivos, con mayúsculas y minúsculas, con la primera letra de cada palabra en mayúscula (por ejemplo, `MyClass`).
 - Los nombres de clases deben contener palabras completas. Evite los acrónimos y abreviaturas (a menos que la abreviatura se utilice mucho más que la forma extendida, como JVM o UML).

- El ejemplo de clase que se muestra en la parte inferior de la diapositiva se describe de la siguiente forma:
 - La clase `Shirt` utiliza un modificador de clase `public`, seguido de la palabra clave `class`, seguida de un nombre de clase `Shirt`.
 - Las llaves se utilizan para delimitar todo el cuerpo del código de la clase `Shirt` y también el cuerpo del código del método `setPrice`.
 - Los paréntesis se utilizan para delimitar el argumento transferido al método `setPrice`. (En una diapositiva posterior verá más de la sintaxis del método).
 - Se utiliza un punto y coma al final de la declaración del campo, `price`.

Requisitos para el archivo de origen

En este curso, desarrollará sus clases para que el código de programación de tecnología Java que escriba para cada clase esté en su propio archivo de texto o archivo de código fuente. En el lenguaje de programación Java, los nombres de archivos de código fuente deben coincidir con el nombre de la clase pública del archivo de código fuente y deben tener una extensión `.java`. Por ejemplo, la clase `Shirt` se debe guardar en un archivo denominado `Shirt.java`.

Prueba

Seleccione la declaración de clase que cumple las instrucciones de nomenclatura de clases.

- a. `class Shirt`
- b. `public Class 501Pants`
- c. `public Shirt`
- d. `public Class Pants`

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Respuesta: a

- La definición de clase va seguida de una llave de apertura ({) que indica el principio de `class_body`, las variables de atributos y los métodos que forman la clase. Las llaves { } alrededor de `class_body` definen dónde empieza y acaba la clase.
- b es incorrecta porque la palabra `class` tiene la inicial en mayúscula.
- c es incorrecta porque la clase no se utiliza en el nombre de clase.
- d es incorrecta porque la palabra `class` tiene la inicial en mayúscula.

Declaraciones y asignaciones de campos

```
public int shirtID = 0;
public String description = "-description required-";
public char colorCode = 'U';
public double price = 0.0;
public int quantityInStock = 0;
```

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

El bloque de declaraciones y asignaciones de campos va después de la primera llave de apertura ({). Normalmente, se configuran todas las variables de atributos para la clase después de esta llave. Observe el punto y coma al final de cada línea de código de este ejemplo.

Comentarios

- **Una sola línea:**

```
public int shirtID = 0; // Default ID for the shirt
public double price = 0.0; // Default price for all shirts

// The color codes are R=Red, B=Blue, G=Green
```

- **Tradicional:**

```
/******
 * Attribute Variable Declaration Section *
 *****/
```

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Debe poner comentarios en cada clase que cree para facilitar la determinación de la acción que realiza el programa. Los comentarios son especialmente importantes en programas más grandes desarrollados por grandes equipos en los que varios programadores tienen que leer el código. Los comentarios ayudan con el mantenimiento de un programa cuando nuevos programadores tienen que determinar la acción que realiza el código.

Se pueden utilizar dos estilos principales de comentarios:

- **Comentarios de una sola línea:** un marcador `//` indica al compilador que ignore todo hasta el final de la línea actual. Muchos programadores también hacen más sencilla la lectura de sus programas utilizando comentarios de una sola línea para comentar la primera y última líneas de cada clase y método. Por ejemplo, la clase `Shirt` contiene un comentario de fin de línea para indicar el final del método de visualización (línea 18):
 - `} // end of display method`

- **Comentarios tradicionales:** una combinación de los caracteres `/*` indica al compilador que ignore lo que aparece en todas las líneas hasta un marcador de terminación de comentario, inclusive `*/`.

- `/******`
- `* Attribute Variable Declaration Section *`
- `*****/`
- Los programadores a menudo utilizan comentarios tradicionales para proporcionar detalles de un gran bloque de código. En grandes programas, puede resultar muy difícil encontrar las llaves de la clase. Al comentar la estructura a la que pertenece cada llave de cierre, la lectura y corrección de errores resulta mucho más sencilla.

Sabía que... Hay un tercer tipo de comentario denominado *comentario de documentación*. Puede utilizar una herramienta de tecnología Java, la herramienta Javadoc, para crear documentación para cualquiera de las clases que utilizarán otros programadores. De hecho, toda la documentación de la biblioteca de clases que se incluye con Java SE JDK se ha creado con la herramienta Javadoc. Los comentarios de documentación deben empezar por una barra inclinada y dos asteriscos (`/**`) y deben terminar con un asterisco y una barra inclinada (`*/`). El ejemplo anterior de un comentario tradicional también sería un comentario de documentación válido.

Temas

- Definición de una clase, identificación de componentes de una clase y uso de variables
- **Análisis de métodos y el uso de un método `main`**
- Identificación de palabras clave
- Prueba y ejecución de un programa Java simple
- Descripción de algunas causas comunes de errores de sintaxis
- Descripción del objetivo y las funciones de un depurador de IDE

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Métodos

- **Sintaxis:**

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

- **Ejemplo:**

```
public void displayInformation() {

    System.out.println("Shirt ID: " + shirtID);
    System.out.println("Shirt description:" + description);
    System.out.println("Color Code: " + colorCode);
    System.out.println("Shirt price: " + price);
    System.out.println("Quantity in stock: " + quantityInStock);

} // end of display method
```

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Los métodos se colocan después de las declaraciones de variables de atributos de una clase. La sintaxis de los métodos es la siguiente:

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

donde:

- `[modifiers]` representa varias palabras clave únicas de tecnología Java que modifican la forma en que se accede a los métodos. Los modificadores son opcionales (se indican con corchetes).
- `return type` indica el tipo de valor (si lo hay) que ejecuta el método. Si el método devuelve un valor, se debe declarar el tipo del valor. Los valores devueltos los puede utilizar el método de llamada. Cualquier método puede devolver al menos un valor. Si el método no devuelve nada, se debe utilizar la palabra clave `void` para `return type`.

- `method_identifier` es el nombre del método.
- `([arguments])` representa una lista de variables cuyos valores se transfieren al método para que los utilice. Los argumentos son opcionales (se indican con corchetes) ya que los métodos no son necesarios para aceptar argumentos. Tenga también en cuenta que los paréntesis no son opcionales. Un método que no acepte argumentos se declara con un juego de paréntesis vacío.
- `method_code_block` es una secuencia de sentencias que realiza el método. Se puede llevar a cabo una gran variedad de tareas en el bloque de código o cuerpo del método. En el ejemplo de código, la clase `Shirt` contiene un método, el método `displayInformation`, que muestra los valores para los atributos de una camisa.

En el método `displayInformation`, ve varias líneas de código que llaman al método `System.out.println`. Este método se utiliza para imprimir una cadena concreta de datos. Utilizará este método en la práctica posterior.

Temas

- Definición de una clase, identificación de componentes de una clase y uso de variables
- Análisis de métodos y el uso de un método main
- **Identificación de palabras clave**
- Prueba y ejecución de un programa Java simple
- Descripción de algunas causas comunes de errores de sintaxis
- Descripción del objetivo y las funciones de un depurador de IDE

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Palabras clave

abstract	default	for	package	synchronized
assert	do	if	private	this
boolean	double	implements	protected	throw
break	else	import	public	throws
byte	enum	instanceof	return	transient
case	extends	int	short	true
catch	false	interface	static	try
char	final	long	strictfp	void
class	finally	native	super	volatile
continue	float	new	switch	while



Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Las palabras clave son palabras especiales reservadas en el lenguaje de programación Java para dar instrucciones al compilador. Las palabras clave no se deben utilizar como identificadores de clases, métodos, variables, etc. En la tabla se incluyen todas las palabras clave de la tecnología Java. `true`, `false` y `null` pueden parecer palabras clave, pero en realidad son literales; no puede utilizarlas como identificadores en los programas.

Temas

- Definición de una clase, identificación de componentes de una clase y uso de variables
- Análisis de métodos y el uso de un método `main`
- Identificación de palabras clave
- **Prueba y ejecución de un programa Java simple**
- Descripción de algunas causas comunes de errores de sintaxis
- Descripción del objetivo y las funciones de un depurador de IDE

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Creación y uso de una clase de prueba

Ejemplo:

```
class ShirtTest {  
  
    public static void main (String[] args) {  
  
        Shirt myShirt;  
        myShirt= new Shirt();  
  
        myShirt.displayInformation();  
  
    }  
}
```

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

La mayoría de las clases que cree a lo largo de este curso no se pueden utilizar (ejecutar y probar) por sí solas. En su lugar, debe ejecutar otra clase para crear una instancia de objeto de la clase para poder probar la clase. En este curso, utilizará una clase de prueba o principal para probar cada una de las clases. El código de la diapositiva es un ejemplo de una clase de prueba para la clase `Shirt`.

A cada clase de prueba de este curso se le debe asignar un nombre para que se pueda reconocer como clase de prueba de una clase concreta escrita. En concreto, cada nombre de clase de prueba está formado por el nombre de la clase de prueba, seguido de la palabra `Test`. Por ejemplo, la clase diseñada para probar la clase `Shirt` se llama `ShirtTest`. Las clases de prueba tienen dos tareas distintas que realizar:

- Proporcionar un punto de inicio, denominado método `main`, para el programa
- Crear una instancia de objeto de la clase y probar sus métodos

Método main

- Método especial que JVM reconoce como punto de inicio de cada programa de tecnología Java que se ejecuta desde una línea de comandos.
- Sintaxis:

```
public static void main (String[] args)
```

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

El método `main` es un método especial que Java Virtual Machine reconoce como punto de inicio de cada programa de tecnología Java que se ejecuta desde la línea de comandos o desde un símbolo del sistema. Cualquier programa que desee ejecutar desde una línea de comandos o un símbolo del sistema debe tener un método `main`.

Sabía que... Muchas de las clases de tecnología Java que crean los ingenieros no se ejecutan en un sistema operativo. ¿Recuerda los applets? Los applets se ejecutan en un explorador web y tienen su propio método de inicio único.

La sintaxis del método `main` es la siguiente:

```
public static void main (String[] args)
```

El método `main` cumple la sintaxis de todos los métodos descrita anteriormente.

En concreto:

- El método `main` contiene dos modificadores necesarios, `public` y `static`.
- El método `main` no devuelve ningún valor, por lo que tiene un tipo de retorno `void`.
- El método `main` tiene un identificador de método (nombre) “main”.
- El método `main` acepta cero o más objetos de tipo `String` (`String[] args`). Esta sintaxis permite introducir valores en la línea de comandos para que los utilice el programa mientras se ejecuta.

Compilación de un programa

1. Vaya al directorio donde están almacenados los archivos de código fuente.
2. Introduzca el siguiente comando para cada archivo `.java` que desee compilar.

- Sintaxis:

```
javac filename
```

- Ejemplo:

```
javac Shirt.java
```

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Esta es una revisión de la actividad de la lección titulada “Introducción a la tecnología Java”. La compilación convierte los archivos de clase que escribe en código de byte que ejecuta Java Virtual Machine. Recuerde las reglas de nomenclatura de los archivos de origen Java. Si un archivo de origen contiene una clase pública, el archivo de origen debe utilizar el mismo nombre que la clase pública, con una extensión `.java`. Por ejemplo, la clase `Shirt` se debe guardar en un archivo denominado `Shirt.java`.

Para compilar los archivos de código fuente `Shirt` y `ShirtTest`:

1. Vaya al directorio en el que están almacenados los archivos de código fuente.
2. Introduzca el siguiente comando para cada archivo `.java` que desee compilar:

```
javac filename
```

Ejemplo:

```
javac Shirt.java
```

Una vez finalizada la compilación y suponiendo que no se ha producido ningún error de compilación, debe tener un nuevo archivo denominado `classname.class` en el directorio para cada archivo de código fuente compilado. Si compila una clase que hace referencia a otros objetos, las clases de dichos objetos también se compilan (si aún no se han compilado). Por ejemplo, si compila el archivo `ShirtTest.java` (que hace referencia a un objeto `Shirt`), puede tener un archivo `Shirt.class` y `ShirtTest.class`.

Ejecución (prueba) de un programa

1. Vaya al directorio en el que están almacenados los archivos de clase.
2. Introduzca lo siguiente para el archivo de clase que contiene el método `main`:

- Sintaxis:

```
java classname
```

- Ejemplo:

```
java ShirtTest
```

- Salida:

```
Shirt ID: 0
Shirt description:-description required-
Color Code: U
Shirt price: 0.0
Quantity in stock: 0
```



Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Una vez compilados correctamente los archivos de código fuente, puede ejecutarlos y probarlos mediante Java Virtual Machine.

Para ejecutar y probar el programa:

1. Vaya al directorio en el que están almacenados los archivos de clase.
2. Introduzca el siguiente comando para el archivo de clase que contiene el método `main`:

```
java classname
```

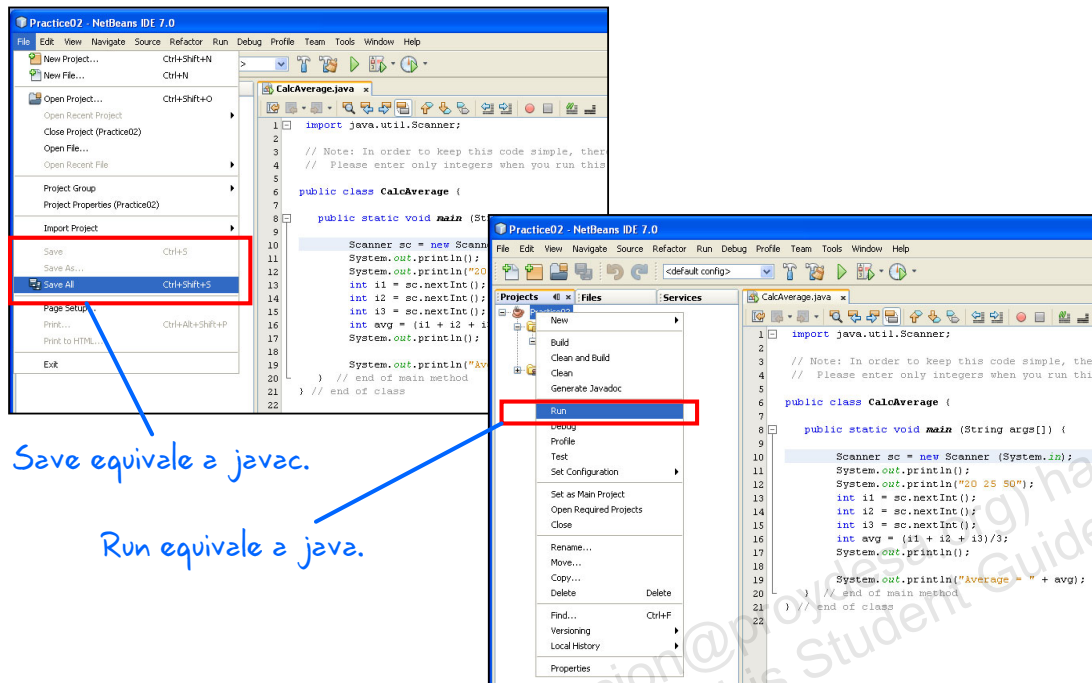
Ejemplo:

```
java ShirtTest
```

Este comando ejecuta la clase `ShirtTest`. Como se ha mencionado anteriormente, la clase `ShirtTest` crea una instancia del objeto `Shirt` mediante la clase `Shirt`. Todos los objetos `Shirt` tienen un método, el método `display`, que imprime los valores de las variables de atributos, como en este ejemplo:

```
Shirt ID: 0
Shirt description:-description required-
Color Code: U
Shirt price: 0.0
Quantity in stock: 0
```

Compilación y ejecución de un programa mediante un IDE



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Save llama al comando `javac <classname(s)>` para todos los archivos `.java` del proyecto. El botón Run File o Run llama al comando `java <classname>`. Asegúrese de observar cualquier indicador de burbuja roja del editor de códigos para localizar errores de sintaxis.

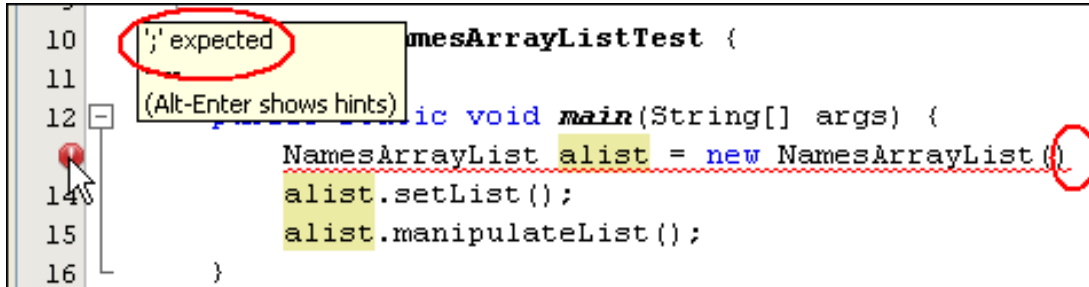
Temas

- Definición de una clase, identificación de componentes de una clase y uso de variables
- Análisis de métodos y el uso de un método `main`
- Identificación de palabras clave
- Prueba y ejecución de un programa Java simple
- **Descripción de algunas causas comunes de errores de sintaxis**
- Descripción del objetivo y las funciones de un depurador de IDE

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Cómo evitar problemas de sintaxis



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

La mayoría de editores Java comprueban la sintaxis del código y muestran alertas mediante iconos y subrayados de color rojo donde hay errores en el código.

Para evitar problemas de sintaxis, asegúrese de llevar a cabo lo siguiente:

- Observe cualquier indicador de burbuja roja del editor de códigos para localizar errores de sintaxis.
- Ponga un punto y coma al final de cada línea donde sea necesario.
- Incluya un número par de símbolos como llaves, corchetes y comillas.

En la captura de pantalla se muestra un error en la línea 13, en la que falta un punto y coma. Si coloca el cursor sobre la burbuja roja, el editor ofrece una sugerencia para corregir el error.

Temas

- Definición de una clase, identificación de componentes de una clase y uso de variables
- Análisis de métodos y el uso de un método `main`
- Identificación de palabras clave
- Prueba y ejecución de un programa Java simple
- Descripción de algunas causas comunes de errores de sintaxis
- Descripción del objetivo y las funciones de un depurador de IDE

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Trabajar con un depurador de IDE

The screenshot shows a Java IDE with a code editor and a variables window. The code editor displays a method `displayShirtInformation()` with several `System.out.println` statements. The variables window is open, showing a table of variables. The variable `this` is highlighted with a red box. The table has columns for Name, Type, and Value.

Name	Type	Value
<Enter new watch>		
this	Shirt	#58
shirtID	int	0
description	String	"-description required-"
colorCode	char	'U'
price	double	0.0
quantityInStock	int	0

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Un depurador permite colocar puntos de ruptura en el código fuente, agregar recursos de comprobación de campos, desplazarse por el código, ejecutar métodos, realizar instantáneas y supervisar la ejecución a medida que se produce. También puede conectar el depurador de NetBeans a un proceso que ya esté en ejecución.

Otras funciones incluyen:

- **Depurador configurable:** puede configurar el comportamiento de ruptura/suspensión, especificar formateadores de variables y omitir métodos y paquetes mediante Step Filters.
- **Ventana Debugging:** la ventana Debugging integra las vistas Sessions, Threads y Call Stack.
- **Puntos de ruptura configurables:** configure estos puntos de ruptura personalizados para que los disparen condiciones y eventos como excepciones no resueltas, una carga de clase o un acceso a variable.
- **Evaluación de expresiones:** evalúe las expresiones de sintaxis Java asignadas a las comprobaciones y los puntos de ruptura condicionales “activos” mientras se desliza por el código.
- **Desplazamiento por expresiones:** vaya a las expresiones individuales de una sentencia.
- **Depuración de varias sesiones:** depure varios procesos al mismo tiempo.
- **HeapWalker:** compruebe referencias a objetos mientras depura un programa.

En la captura de pantalla, puede ver un programa en mitad de una sesión de depuración. La flecha del panel izquierdo indica que el IDE se está desplazando por el código y esta es la siguiente línea que se ejecutará. En la ventana Variables de la parte inferior de la pantalla, puede ver los campos de la clase que se está ejecutando actualmente (a la que se hace referencia con la palabra clave `this`). Durante una sesión de depuración, puede cambiar los valores de estos campos para intentar distintos supuestos. Esto resulta útil para solucionar problemas lógicos.

Resumen

En esta lección, debe haber aprendido lo siguiente:

- Definir una clase
- Identificar los componentes de una clase
- Explicar el término *objeto*
- Describir el objetivo de una variable
- Analizar métodos y describir cómo utilizar un método `main`
- Describir los elementos que componen una clase Java, como declaraciones, valores de retorno y el uso correcto de los corchetes y las llaves
- Identificar palabras clave y describir su objetivo
- Probar y ejecutar un programa simple
- Describir algunas causas comunes de errores de sintaxis
- Describir el objetivo y las funciones de un depurador de IDE



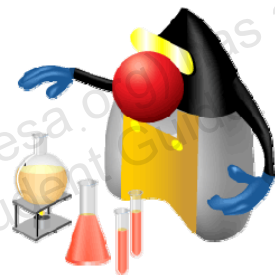
ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Visión general de la práctica 4-1: Visualización y adición de código en un programa Java existente

En esta práctica, se proporciona un programa Java terminado. Durante la práctica:

- Abrirá el programa Java.
- Examinará las líneas de código.
- Modificará el programa.
- Compilará el programa.
- Probará el programa mediante su ejecución.

**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Visión general de la práctica 4-2: Creación y compilación de una clase Java

En esta práctica, creará una clase Java y la compilará.
También creará otra clase Java para probar la clase anterior.

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Visión general de la práctica 4-3: Exploración del depurador

En esta práctica, depurará el programa `ShirtTest` mediante el depurador de NetBeans. Durante la práctica:

- Definirá un punto de ruptura.
- Examinará y modificará los valores de campos.
- Utilizará un desplazamiento.

The Oracle logo, consisting of the word "ORACLE" in a bold, sans-serif font, with a registered trademark symbol (®) to its upper right.

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Declaración, inicialización y uso de variables

ORACLE®

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos

Al finalizar esta lección, debería estar capacitado para:

- Identificar los usos de las variables y definir la sintaxis de una variable
- Enumerar los ocho tipos de dato primitivos del lenguaje de programación Java
- Declarar, inicializar y utilizar variables y constantes según las instrucciones del lenguaje de programación Java y los estándares de codificación
- Modificar valores de variables mediante operadores
- Utilizar la ampliación y la conversión de tipo



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Importancia

- Una variable hace referencia a algo que puede cambiar. Las variables pueden contener un valor de un juego de valores. ¿Dónde ha visto variables con anterioridad?
- ¿Qué tipos de dato cree que pueden contener variables?

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Temas

- Identificación de los usos de las variables y definición de la sintaxis de una variable
- Enumeración de los ocho tipos de dato primitivos del lenguaje de programación Java
- Declaración, inicialización y uso de variables y constantes
- Modificación de valores de variables mediante operadores
- Uso de ampliación y conversión de tipo

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Identificación del uso y la sintaxis de las variables

Ejemplo:

```
public class Shirt {  
  
    public int shirtID = 0; // Default ID for the shirt  
  
    public String description = "-description required-"; // default  
    // The color codes are R=Red, B=Blue, G=Green, U=Unset  
    public char colorCode = 'U';  
  
    public double price = 0.0; // Default price for all shirts  
  
    public int quantityInStock = 0; // Default quantity for all shirts  
  
    // This method displays the values for an item  
    public void displayInformation() {  
  
        System.out.println("Shirt ID: " + shirtID);  
    }  
}
```

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Las variables se utilizan para almacenar y recuperar datos del programa. Los objetos almacenan sus estados individuales en campos. Los campos también se denominan *variables de instancia* porque sus valores son únicos para cada instancia individual de una clase. El ejemplo de código muestra una clase `Shirt` que declara varios campos no estáticos (como `price`, `shirtID` y `colorCode` de la clase `Shirt`). Cuando se instancia un objeto de una clase, estas variables contienen datos específicos de una instancia de objeto concreta de la clase. Por ejemplo, una instancia de la clase `Shirt` puede tener el valor 7 asignado al campo no estático `quantityInStock`, mientras que otra instancia de la clase `Shirt` puede tener el valor 100 asignado al campo no estático `quantityInStock`.

Identificación del uso y la sintaxis de las variables

Ejemplo:

```
public void displayDescription {  
    String displayString = "";  
    displayString = "Shirt description: " + description;  
    System.out.println(displayString);  
}
```

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Los programas también tienen variables definidas en métodos. Estas variables se denominan *variables locales* porque solo están disponibles de forma local en el método en el que se declaran.

Nota: en este curso, los términos *variables* o *campos* se utilizan para hacer referencia a variables. Si la situación lo necesita, se utilizará *variable local* cuando corresponda.

Usos de las variables

- Contener datos únicos para una instancia de objeto
- Asignar el valor de una variable a otra
- Representar valores en una expresión matemática
- Imprimir los valores en la pantalla
- Contener referencias a otros objetos



Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Las variables se utilizan ampliamente en el lenguaje de programación Java para tareas como, por ejemplo:

- Contener datos de atributos únicos para una instancia de objeto (como ha visto con las variables `price` e `ID`)
- Asignar el valor de una variable a otra
- Representar valores en una expresión matemática
- Mostrar los valores en la pantalla. Por ejemplo, la clase `Shirt` utiliza las variables `price` e `ID` para imprimir los valores de precio e identificador de la camisa:

```
System.out.println("Shirt price: " + price);
System.out.println("Shirt ID: " + shirtID);
```
- Contener referencias a otros objetos

Declaración e inicialización de variables

- Sintaxis (campos):

```
[modifiers] type identifier [= value];
```

- Sintaxis (variables locales):

```
type identifier [= value];
```

- Ejemplos:

```
public int shirtID = 0;
public String description = "-description required-";
public char colorCode = 'U';
public double price = 0.0;
public int quantityInStock = 0;
```



Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

La declaración e inicialización de variables de atributos sigue la misma sintaxis general. La sintaxis para declarar e inicializar campos es la siguiente:

```
[modifiers] type identifier [= value];
```

La sintaxis para inicializar una variable en un método es la siguiente:

```
identifier = value;
```

La sintaxis para declarar e inicializar una variable en un método es la siguiente:

```
type identifier [= value];
```

donde:

- `[modifiers]` representa varias palabras clave especiales de tecnología Java, como `public` y `private`, que modifican el acceso que otro código tiene a un campo. Los modificadores son opcionales (se indican con corchetes). Por ahora, todos los campos que cree deben tener un modificador `public`.
- `type` representa el tipo de información o datos que contiene la variable. Algunas variables contienen caracteres, otras contienen números y otras son booleanos y solo pueden contener uno de dos valores. Todas las variables deben tener asignado un tipo para indicar el tipo de información que contienen.

Nota: no utilice modificadores con variables locales (variables declaradas en métodos).

- `identifier` es el nombre asignado a la variable que es de tipo `type`.
- `value` es el valor que desea asignar a la variable. El valor es opcional ya que no necesita asignar un valor a una variable en el momento que declara la variable.

A continuación, se muestran las declaraciones de los campos de la clase `Shirt`:

```
public int shirtID = 0;
public String description = "-description required-";
public char colorCode = 'U';
public double price = 0.0;
public int quantityInStock = 0;
```

Temas

- Identificación de los usos de las variables y definición de la sintaxis de una variable
- **Enumeración de los ocho tipos de dato primitivos del lenguaje de programación Java**
- Declaración, inicialización y uso de variables y constantes
- Modificación de valores de variables mediante operadores
- Uso de ampliación y conversión de tipo

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Descripción de tipos de dato primitivos

- Tipos integrales (`byte`, `short`, `int` y `long`)
- Tipos de coma flotante (`float` y `double`)
- Tipo textual (`char`)
- Tipo lógico (`boolean`)

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Muchos de los valores de los programas de tecnología Java se almacenan como tipos de dato primitivos. En la diapositiva se muestran los ocho tipos primitivos incorporados en el lenguaje de programación Java.

Tipos primitivos integrales

Tipo	Longitud	Rango	Ejemplos de valores literales permitidos
byte	8 bits	De -2^7 a $2^7 - 1$ (de -128 a 127, o 256 posibles valores)	2 -114 0b10 (número binario)
short	16 bits	De -2^{15} a $2^{15} - 1$ (de -32.768 a 32.767, o 65.535 posibles valores)	2 -32699
int (tipo por defecto para literales integrales)	32 bits	De -2^{31} a $2^{31} - 1$ (de -2.147.483.648 a 2.147.483.647, o 4.294.967.296 posibles valores)	2 147334778 123_456_678

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Hay cuatro tipos primitivos integrales en el lenguaje de programación Java, identificados con las palabras clave `byte`, `short`, `int` y `long`. Estos tipos almacenan números que no tienen comas decimales. Si necesita almacenar las edades de personas, por ejemplo, servirá una variable de tipo `byte` ya que los tipos `byte` pueden aceptar valores de ese rango. Al especificar un valor literal para un tipo `long`, ponga una `L` mayúscula a la derecha del valor para indicar explícitamente que es un tipo `long`. El compilador asume que los literales integrales son de tipo `int` a menos que especifique lo contrario mediante una `L` que indique el tipo `long`.

Una nueva función de SE 7 permite expresar cualquiera de los tipos integrales como binarios (ceros y unos). Por ejemplo, una expresión binaria del número 2 se muestra como un valor permitido del tipo integral `byte`. El valor binario es `0b10`. Observe que este valor empieza por `0b` (es decir, cero seguido de una letra `B` minúscula o mayúscula). Esto indica al compilador que, a continuación, viene un valor binario.

Otra nueva función de SE 7 se puede ver en la fila `int`. La posibilidad de incluir caracteres de subrayado en un número `int` largo ayuda a la lectura del código. Por ejemplo, puede utilizar esto para facilitar la lectura de un número integral largo mediante la sustitución de los caracteres de subrayado con comas. El uso del carácter de subrayado no tiene ningún efecto en el valor numérico de `int` ni aparece si la variable se imprime en la pantalla.

Tipos primitivos integrales

Tipo	Longitud	Rango	Ejemplos de valores literales permitidos
long	64 bits	De -2^{63} a $2^{63} - 1$ (de $-9.223.372.036.854.775.808$ a $9.223.372.036.854.775.807$, o $18.446.744.073.709.551,616$ posibles valores)	2 -2036854775808L 1L

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

La clase `Shirt` contiene dos atributos de tipo `int` para incluir los valores de `shirtID` y la cantidad en stock, mientras que los valores literales se utilizan para proporcionar un valor de inicio por defecto de cero (0) para cada uno.

```
public int shirtID = 0; // Default ID for the shirt
public int quantityInStock = 0; // Default quantity for all shirts
```

Nota: el único motivo para utilizar los tipos `byte` y `short` en programas es ahorrar consumo de memoria. Puesto que la mayoría de computadoras de escritorio modernas tienen mucha memoria, la mayoría de programadores de aplicaciones de escritorio no utilizan los tipos `byte` y `short`. En este curso se utilizan principalmente los tipos `int` y `long` en los ejemplos.

Tipos primitivos de coma flotante

Tipo	Longitud Float	Ejemplos de valores literales permitidos
float	32 bits	99F -327456,99.01F 4.2E6F (notación de ingeniería para $4,2 * 10^6$)
double (tipo por defecto de los literales de coma flotante)	64 bits	-1111 2.1E12 99970132745699.999

```
public double price = 0.0; // Default price for all shirts
```

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Hay dos tipos de números de coma flotante: `float` y `double`. Estos tipos se utilizan para almacenar números con valores a la derecha del punto decimal, como 12.24 o 3.14159. Al especificar un valor literal para un tipo `float`, ponga una `F` mayúscula (`float`) a la derecha del valor para indicar explícitamente que es un tipo `float` y no un tipo `double`.

Se asume que los valores literales para tipos de coma flotante son de tipo `double` a menos que especifique lo contrario mediante la `F` que indique el tipo `float`. La clase `Shirt` muestra el uso de un valor literal `double` para especificar el valor por defecto del precio:

```
public double price = 0.0; // Default price for all shirts
```

Nota: utilice el tipo `double` cuando sea necesario un rango o precisión mayor.

Tipo primitivo textual

- El único tipo de dato textual primitivo es `char`.
- Se utiliza para un único carácter (16 bits).
- Ejemplo:
 - `public char colorCode = 'U';`

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Otro tipo de dato que se utiliza para almacenar y manipular datos es la información de un único carácter. El tipo primitivo utilizado para almacenar un único carácter (como `y`) es `char`, que tiene un tamaño de 16 bits. La clase `Shirt` muestra el uso de un valor literal textual para especificar el valor por defecto de `colorCode`:

```
public char colorCode = 'U';
```

Al asignar un valor literal a una variable `char`, como `t`, debe utilizar comillas simples alrededor del carácter: `'t'`. El uso de comillas simples alrededor del carácter aclara al compilador que la `t` es solo el valor literal `t`, en lugar de una variable `t` que representa otro valor.

El tipo `char` no almacena el carácter real escrito, como la `t` mostrada. La representación `char` se reduce a una serie de bits que corresponde a un carácter. Las asignaciones de carácter de número se configuran en el juego de caracteres que utiliza el lenguaje de programación.

Sabía que... Muchos lenguajes informáticos utilizan ASCII (American Standard Code for Information Interchange), un juego de caracteres de 8 bits que tiene una entrada para cada carácter inglés, signo de puntuación, número, etc.

El lenguaje de programación Java utiliza un juego de caracteres de 16 bits denominado Unicode que puede almacenar todos los caracteres visualizables necesarios para la gran mayoría de idiomas utilizados en la actualidad. Por lo tanto, los programas se pueden escribir de forma que funcionen correctamente y se muestren en el idioma correcto de la mayoría de países. Unicode contiene un subjuego de ASCII (los primeros 128 caracteres).

Tipo primitivo lógico

- El único tipo de dato es `boolean`.
- Solo puede almacenar `true` o `false`.
- Contiene el resultado de una expresión que se evalúa en `true` o `false`.



Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Los programas informáticos a menudo deben tomar decisiones. El resultado de una decisión, si la sentencia en el programa es `true` o `false`, se puede guardar en variables booleanas. Las variables de tipo `boolean` solo pueden almacenar:

- Los literales del lenguaje de programación Java `true` o `false`.
- El resultado de una expresión que se evalúa solo en `true` o `false`. Por ejemplo, si la respuesta a la variable es igual a 42, la expresión `"if answer < 42"` se evalúa en un resultado `false`.

Temas

- Identificación de los usos de las variables y definición de la sintaxis de una variable
- Enumeración de los ocho tipos de dato primitivos del lenguaje de programación Java
- **Declaración, inicialización y uso de variables y constantes**
- Modificación de valores de variables mediante operadores
- Uso de ampliación y conversión de tipo

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Asignación de nombres a variables

Reglas:

- Los identificadores de variables deben empezar por una letra mayúscula o minúscula, un carácter de subrayado (`_`) o un signo de dólar (`$`).
- Los identificadores de variables no pueden contener puntuación, espacios ni guiones.
- No se pueden utilizar las palabras clave de la tecnología Java.



Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Al igual que con una clase o método, debe asignar un identificador o nombre a cada variable del programa. Recuerde que el objetivo de la variable es actuar como mecanismo para almacenar y recuperar valores. Por lo tanto, debe hacer que los identificadores de variables sean simples pero descriptivos. Por ejemplo, si almacena el valor de un ID de elemento, puede asignar a la variable el nombre `myID`, `itemID`, `itemNumber` o cualquier otro que le aclare el uso de la variable a usted y a otras personas que lean el programa.

Sabía que... Muchos programadores siguen la convención de utilizar la primera letra del tipo como identificador: `int i`, `float f`, etc. Esta convención es aceptable para programas pequeños que son fáciles de descifrar, pero en general debe utilizar identificadores más descriptivos.

Asignación de nombres a variables

Instrucciones:

- Empezar cada variable por una letra minúscula. Las siguientes palabras deben tener la inicial mayúscula (por ejemplo, `myVariable`).
- Seleccionar nombres que sean nemotécnicos y que indiquen al observador casual la intención de la variable.

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Asignación de un valor a una variable

- Ejemplo:
 - `double price = 12.99;`
- Ejemplo (booleano):
 - `boolean isOpen = false;`



Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Puede asignar un valor a una variable al declarar la variable o asignar la variable más tarde. Para asignar un valor a una variable durante la declaración, agregue un signo igual (=) después de la declaración, seguido del valor que se va a asignar. Por ejemplo, al campo de precio de la clase `Shirt` se puede asignar el valor `12.99` como precio para un objeto `Shirt`.

```
double price = 12.99;
```

Un ejemplo de declaración y asignación de variable booleana es el siguiente:

```
boolean isOpen = false;
```

El operador = asigna el valor de la derecha al elemento de la izquierda. El operador = se debe leer como “está asignado a”. En el ejemplo anterior, puede decir “12.99 está asignado a precio”. Los operadores, como el operador de asignación (=), se explican más adelante en este curso.

Nota: los campos se inicializan automáticamente: los tipos integrales se definen en 0, los tipos de coma flotante se definen en 0.0, el tipo `char` se define en `\u0000` y el tipo `boolean` se define en `false`. Sin embargo, debe inicializar explícitamente los campos para que otras personas puedan leer el código. Las variables locales (declaradas en un método) se deben inicializar explícitamente antes de su uso.

Declaración e inicialización de varias variables en una línea de código

- Sintaxis:
 - `type identifier = value [, identifier = value];`
- Ejemplo:
 - `double price = 0.0, wholesalePrice = 0.0;`



Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Puede declarar una o más variables en la misma línea de código, pero solo si todas son del mismo tipo. La sintaxis para declarar varias variables en una línea de código es la siguiente:

```
type identifier = value [, identifier = value];
```

Por lo tanto, si hay precios de minorista y de mayorista independientes en la clase `Shirt`, se pueden declarar de la siguiente forma:

```
double price = 0.0, wholesalePrice = 0.0;
```

Métodos adicionales para declarar variables y asignar valores a variables

- Asignación de valores literales:
 - `int ID = 0;`
 - `float pi = 3.14F;`
 - `char myChar = 'G';`
 - `boolean isOpen = false;`
- Asignación del valor de una variable a otra:
 - `int ID = 0;`
 - `int saleID = ID;`

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

También puede asignar valores a variables mediante varios enfoques distintos.

- Asignación de valores literales directamente a variables (como se ha descrito en esta lección):

```
int ID = 0;
float pi = 3.14F;
char myChar = 'G';
boolean isOpen = false;
```

- Asignación del valor de una variable a otra:

```
int ID = 0;
int saleID = ID;
```

La primera línea de código crea un entero denominado `ID` y lo utiliza para almacenar el número 0. La segunda línea de código crea otro entero denominado `saleID` y lo utiliza para almacenar el mismo valor como `ID` (0). Si el contenido de `ID` se cambia posteriormente, el contenido de `saleID` no cambia automáticamente. Incluso aunque los dos enteros tienen actualmente el mismo valor, se pueden cambiar de forma independiente en un programa.

Métodos adicionales para declarar variables y asignar valores a variables

- Asignación del resultado de una expresión a variables integrales, de coma flotante o booleanas:
 - `float numberOrdered = 908.5F;`
 - `float casePrice = 19.99F;`
 - `float price = (casePrice * numberOrdered);`
 - `int hour = 12;`
 - `boolean isOpen = (hour > 8);`
- Asignación del valor de retorno de una llamada a método a una variable



Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

- Asignación del resultado de una expresión a variables de tipo integral, de coma flotante o booleano:

En las siguientes líneas de código, el resultado de todo lo que está a la derecha del operador = se asigna a la variable situada a la izquierda del operador =.

```
float numberOrdered = 908.5F;
float casePrice = 19.99F;
float price = (casePrice * numberOrdered);
int hour = 12;
boolean isOpen = (hour > 8);
```

- Asignación del valor de retorno de una llamada a método a una variable (este enfoque se describe posteriormente en el curso).

Constantes

- Variable (puede cambiar):
 - `double salesTax = 6.25;`
- Constante (no puede cambiar):
 - `final int NUMBER_OF_MONTHS = 12;`
- Instrucciones: las constantes deben ir en mayúscula, con las palabras separadas con un carácter de subrayado (_).

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En esta lección, se han explicado las variables que tienen valores que se pueden cambiar. En esta sección, aprenderá a utilizar constantes para representar valores que no pueden cambiar.

Supongamos que está escribiendo parte de una aplicación de programación y necesita hacer referencia al número de meses de un año. Convierta la variable en una constante mediante el uso de la palabra clave `final` para informar al compilador de que no desea que se cambie el valor de la variable una vez inicializada. Asimismo, por convención, asigne al identificador de la constante un nombre con todas las letras en mayúscula y con caracteres de subrayado para separar las palabras, de forma que sea fácil determinar que es una constante:

```
final int NUMBER_OF_MONTHS = 12;
```

Cualquier valor que tienda a cambiar en rara ocasión, si lo llega a hacer, es un buen candidato para ser una variable constante (por ejemplo, `MAX_COUNT`, `PI`, etc.).

Si alguien intenta cambiar el valor de una constante después de que ya se haya asignado un valor, el compilador mostrará un mensaje de error. Si modifica el código para proporcionar otro valor para la constante, tiene que volver a compilar el programa.

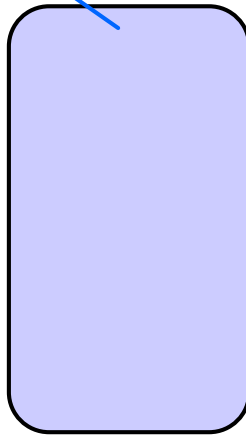
Instrucciones para asignar nombres a constantes

Debe asignar nombres a constantes para que se puedan identificar fácilmente. Por norma general, las constantes deben ir en mayúscula, con las palabras separadas con un carácter de subrayado (_).

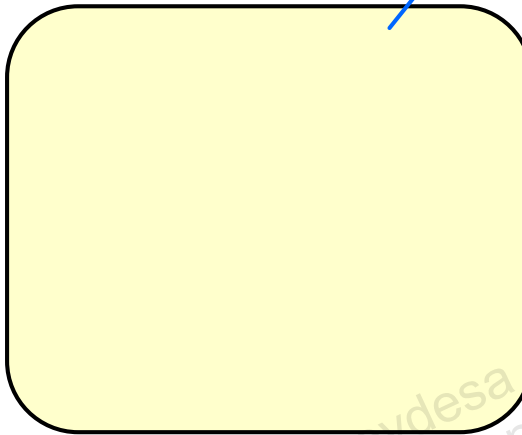
Almacenamiento de primitivos y constantes en memoria

Variable local declarada en un método

Objetos con campos



Memoria de pila



Memoria de montón

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Cuando utiliza un valor literal o crea una variable o constante y le asigna un valor, el valor se almacena en la memoria de la computadora.

En la figura se muestra que las variables locales se almacenan de forma separada (en la pila) de los campos (en el montón). Los objetos y sus campos y métodos se suelen almacenar en la memoria de montón. La memoria de montón se compone de fragmentos de memoria asignados dinámicamente que contienen información utilizada para incluir objetos (incluidos sus campos y métodos) mientras los necesita el programa. Otras variables se suelen almacenar en la memoria de pila. La memoria de pila almacena elementos que se utilizan solo durante un breve período de tiempo (menor que la vida de un objeto), como las variables declaradas en un método.

Prueba

La declaración de variable `public int myInteger=10;` cumple la sintaxis de declaración e inicialización de variables.

- a. Verdadero
- b. Falso

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Respuesta: a

Temas

- Identificación de los usos de las variables y definición de la sintaxis de una variable
- Enumeración de los ocho tipos de dato primitivos del lenguaje de programación Java
- Declaración, inicialización y uso de variables y constantes
- **Modificación de valores de variables mediante operadores**
- Uso de ampliación y conversión de tipo

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Operadores matemáticos estándar

Objetivo	Operador	Ejemplo	Comentarios
Suma	+	sum = num1 + num2; Si num1 es 10 y num2 es 2, sum es 12.	
Resta	-	diff = num1 - num2; Si num1 es 10 y num2 es 2, diff es 8.	
Multiplicación	*	prod = num1 * num2; Si num1 es 10 y num2 es 2, prod es 20.	
División	/	quot = num1 / num2; Si num1 es 31 y num2 es 6, quot es 5.	La división devuelve un valor entero (sin resto).

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Operadores matemáticos estándar

Objetivo	Operador	Ejemplo	Comentarios
Resto	%	<code>mod = num1 % num2;</code> Si num1 es 31 y num2 es 6, mod es 1.	El resto busca el resto del primer número dividido entre el segundo número. <div style="text-align: right;"> 5 R 1 </div> <div style="text-align: right;"> $\begin{array}{r} 6 \overline{) 31} \\ \underline{30} \\ 1 \end{array}$ </div> El resto siempre da una respuesta con el mismo signo como primer operando.

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Los programas realizan muchos cálculos matemáticos, desde simples hasta complejos. Los operadores aritméticos permiten especificar cómo se deben evaluar y combinar los valores numéricos en las variables. Los operadores matemáticos estándar (a menudo llamados *operadores binarios*) utilizados en el lenguaje de programación Java se muestran en las tablas de esta sección.

Nota: el operador % se conoce como módulo.

Operadores de aumento y disminución (++ y --)

Forma extendida:

```
age = age + 1;
```

o bien

```
count = count - 1;
```



Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Un requisito común en los programas es sumar o restar 1 al valor de una variable. Para ello, puede utilizar el operador + de la siguiente forma:

```
age = age + 1;
```

Operadores de aumento y disminución (++ y --)

Forma breve:

Operador	Objetivo	Ejemplo	Notas
++	Aumento previo (++ <i>variable</i>)	<pre>int i = 6; int j = ++i; i is 7, j is 7</pre>	
	Aumento posterior (<i>variable</i> ++)	<pre>int i = 6; int j = i++; i is 7, j is 6</pre>	El valor <i>i</i> se asigna a <i>j</i> antes de aumentar <i>i</i> . Por lo tanto, a <i>j</i> se asigna 6.

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Sin embargo, el aumento o la disminución en 1 es una acción tan común que hay operadores unarios específicos para ello: los operadores de aumento (++) y disminución (--). Estos operadores pueden ir antes (aumento previo y disminución previa) o después (aumento posterior y disminución posterior) de una variable.

La línea de código de la diapositiva anterior, en la que la edad se aumenta en 1, también se puede escribir de la siguiente forma:

```
age++; o ++age;
```

Operadores de aumento y disminución (++ y --)

Operador	Objetivo	Ejemplo	Notas
--	Disminución previa (-- <i>variable</i>)	<pre>int i = 6; int j = --i; i is 5, j is 5</pre>	
	Disminución posterior (<i>variable</i> --)	<pre>int i = 6; int j = i--; i is 5, j is 6</pre>	El valor <i>i</i> se asigna a <i>j</i> antes de disminuir <i>i</i> . Por lo tanto, a <i>j</i> se asigna 6.

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Utilice estos operadores en una expresión con cuidado. Con la forma de prefijo, la operación (aumento o disminución) se aplica antes que cualquier cálculo o asignación siguiente. Con la forma de sufijo, la operación se aplica después de los cálculos u operaciones siguientes, de forma que se utiliza el valor original, y no el valor actualizado, en los cálculos o asignaciones siguientes. En la tabla se muestran los operadores de aumento y disminución.

Operadores de aumento y disminución (++ y --)

Ejemplos:

```
int count=15;
int a, b, c, d;
a = count++;
b = count;
c = ++count;
d = count;
System.out.println(a + ", " + b + ", " + c + ", " + d);
```



Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En el ejemplo de la diapositiva se muestra el uso básico de los operadores de aumento y disminución:

```
int count=15;
int a, b, c, d;
a = count++;
b = count;
c = ++count;
d = count;
System.out.println(a + ", " + b + ", " + c + ", " + d);
```

El resultado de este fragmento de código es el siguiente:

15, 16, 17, 17

Análisis: ¿Cuál es el resultado del código siguiente?

```
int i = 16;
System.out.println(++i + " " + i++ + " " + i);
```


Prioridad de operadores

A continuación se presenta un ejemplo de la necesidad de reglas de prioridad.

¿La respuesta del siguiente problema es 34 o 9?

$$c = 25 - 5 * 4 / 2 - 10 + 4;$$
ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Prioridad de operadores

Reglas de prioridad:

1. Operadores delimitados por un par de paréntesis
2. Operadores de aumento y disminución
3. Operadores de multiplicación y división, evaluados de izquierda a derecha
4. Operadores de suma y resta, evaluados de izquierda a derecha



Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En una sentencia matemática compleja con varios operadores en la misma línea, ¿cómo selecciona la computadora el operador que debe utilizar primero? Para realizar operaciones matemáticas consistentes, el lenguaje de programación Java sigue las reglas matemáticas estándar en cuanto a la prioridad de los operadores. Los operadores se procesan en el siguiente orden:

1. Operadores delimitados por un par de paréntesis
2. Operadores de aumento y disminución
3. Operadores de multiplicación y división, evaluados de izquierda a derecha
4. Operadores de suma y resta, evaluados de izquierda a derecha

Si en una sentencia aparecen sucesivamente operadores matemáticos estándar con la misma prioridad, los operadores se evalúan de izquierda a derecha.

Ejemplo de la necesidad de reglas de prioridad

En el siguiente ejemplo se muestra la necesidad de establecer la prioridad de los operadores:

$$c = 25 - 5 * 4 / 2 - 10 + 4;$$

En este ejemplo, no queda clara la intención del autor. El resultado se puede evaluar de dos formas:

- El resultado de la expresión cuando se evalúa estrictamente de izquierda a derecha: 34
- El resultado real de la expresión cuando se evalúa según las reglas de prioridad, indicadas por los paréntesis: 9

$$c = 25 - ((5 * 4) / 2) - 10 + 4;$$

Uso de paréntesis

Ejemplos:

```
c = ((25 - 5) * 4) / (2 - 10) + 4;  
c = ((20 * 4) / (2 - 10)) + 4;  
c = (80 / (2 - 10)) + 4;  
c = (80 / -8) + 4;  
c = -10 + 4;  
c = -6;
```

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

La expresión se evaluará automáticamente con las reglas de prioridad. Sin embargo, debe utilizar paréntesis para proporcionar la estructura que desea:

```
c = ((25 - 5) * 4) / (2 - 10) + 4;  
c = ((20 * 4) / (2 - 10)) + 4;  
c = (80 / (2 - 10)) + 4;  
c = (80 / -8) + 4;  
c = -10 + 4;  
c = -6;
```

Temas

- Identificación de los usos de las variables y definición de la sintaxis de una variable
- Enumeración de los ocho tipos de dato primitivos del lenguaje de programación Java
- Declaración, inicialización y uso de variables y constantes
- Modificación de valores de variables mediante operadores
- **Uso de ampliación y conversión de tipo**

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de ampliación y conversión de tipo

- Ejemplo de un posible problema:

```
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
```

- Ejemplo de una posible solución:

```
int num1 = 53;
int num2 = 47;
int num3;
num3 = (num1 + num2);
```

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

La asignación de una variable o expresión a otra variable puede producir que no coincidan los tipos de dato del cálculo y la ubicación de almacenamiento que utiliza para guardar el resultado. En concreto, el compilador reconocerá que se perderá la precisión y no permitirá compilar el programa, o bien el resultado será incorrecto. Para corregir este problema, los tipos de variables se tienen que ampliar a un tipo de tamaño mayor o convertir el tipo en un tipo de tamaño menor.

Por ejemplo, considere la siguiente asignación:

```
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
```

Este código debe funcionar, porque `byte`, aunque es menor que `int`, es lo suficientemente grande para almacenar un valor 100. Sin embargo, el compilador no realizará esta asignación y, en su lugar, emitirá un error de “posible pérdida de precisión” porque un valor `byte` es menor que un valor `int`. Para corregir este problema, puede convertir el tipo del tipo de dato de la derecha para que coincida con el tipo de dato de la izquierda, o bien declarar la variable de la izquierda (`num3`) para que sea un tipo de dato mayor, como `int`.

Para corregir este problema, se debe cambiar `num3` a `int`:

```
int num1 = 53;  
int num2 = 47;  
int num3;  
num3 = (num1 + num2);
```

Ampliación

- Ampliaciones automáticas:
 - Si asigna un tipo más pequeño a un tipo mayor.
 - Si asigna un tipo integral a un tipo de coma flotante.
- Ejemplo de ampliaciones automáticas:
`long big = 6;`



Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En algunas circunstancias, el compilador cambia el tipo de una variable a un tipo que soporta un valor de tamaño mayor. Esta acción se denomina *ampliación*. Algunas ampliaciones las realiza automáticamente el compilador si los datos no se pierden al hacerlo. Las ampliaciones se producen:

- Si asigna un tipo más pequeño (a la derecha de =) a un tipo mayor (a la izquierda de =).
- Si asigna un tipo integral a un tipo de coma flotante (porque no hay ningún decimal que se pueda perder).

El siguiente ejemplo contiene un literal (`int`) que se ampliará automáticamente a otro tipo (`long`) antes de que se asigne el valor (6) a la variable (`big` del tipo `long`).

```
long big = 6;
```

Puesto que 6 es un tipo `int`, la ampliación funciona porque el valor `int` se convierte a un valor `long`.

Atención: antes de que se asigne a una variable, el resultado de una ecuación se coloca en una ubicación temporal de la memoria. El tamaño de la ubicación siempre es igual al tamaño de un tipo `int` o al tamaño del tipo de dato mayor utilizado en la expresión o sentencia. Por ejemplo, si la ecuación multiplica dos tipos `int`, el tamaño del contenedor será un tipo `int` en cuanto al tamaño o de 32 bits.

Si los dos valores que multiplican producen un valor que está más allá del ámbito de un tipo `int`, (como $55555 * 66666 = 3.703.629.630$, que es demasiado grande para encajar en un tipo `int`), el valor `int` se debe truncar para que encaje el resultado en la ubicación temporal de la memoria. Este cálculo finalmente produce una respuesta incorrecta porque la variable de la respuesta recibe un valor truncado (independientemente del tipo utilizado para la respuesta).

Para solucionar este problema, defina al menos una de las variables de la ecuación en el tipo `long` para asegurar el mayor tamaño de contenedor temporal posible.

Conversión de tipo

- **Sintaxis:**

```
identifier = (target_type) value
```

- **Ejemplo de un posible problema:**

```
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
```

- **Ejemplo de una posible solución:**

```
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 = (byte)(num1 + num2); // no data loss
```

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

La conversión de tipo disminuye el rango de un valor. Para ello, lo corta literalmente hasta un tamaño menor y cambia el tipo del valor (por ejemplo, convierte un valor `long` en un valor `int`). Esto permite utilizar métodos que acepten solo determinados tipos como argumentos, de forma que pueda asignar valores a una variable de un tipo de dato menor o de forma que pueda ahorrar memoria. Ponga `target_type` (el tipo al que se va a convertir el tipo) entre paréntesis delante del elemento cuyo tipo está cambiando. La sintaxis para convertir el tipo de un valor es la siguiente:

```
identifier = (target_type) value
```

donde:

- `identifier` es el nombre asignado a la variable.
- `value` es el valor que desea asignar al identificador.
- (`target_type`) es el tipo al que desea convertir el valor. Tenga en cuenta que `target_type` debe estar entre paréntesis.

Por ejemplo, considere la siguiente asignación:

```
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
```

El error del compilador se corrige mediante la conversión del tipo del resultado a `byte`.

```
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 = (byte)(num1 + num2); // no data loss
```

Atención: utilice la conversión de tipo con cuidado. Por ejemplo, si se utilizaron números mayores para `num1` y `num2`, la conversión de tipo a `byte` truncará parte de los datos, lo que producirá una respuesta incorrecta.

Conversión de tipo

Ejemplos:

```
int myInt;
long myLong = 99L;
myInt = (int) (myLong); // No data loss, only zeroes.
                        // A much larger number would
                        // result in data loss.

int myInt;
long myLong = 123987654321L;
myInt = (int) (myLong); // Number is "chopped"
```

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Otros posibles problemas son los siguientes:

```
int myInt;
long myLong = 99L;
myInt = (int) (myLong); // No data loss, only zeroes.
// A much larger number would
// result in data loss.

int myInt;
long myLong = 123987654321L;
myInt = (int) (myLong); // Number is "chopped"
```

Si convierte el tipo de un valor `float` o `double` con una parte de fracción a un tipo entero como `int`, se perderán todos los valores decimales. Sin embargo, este método de conversión de tipo a veces resulta útil si desea truncar el número para reducirlo al número entero (por ejemplo, 51,9 se convierte en 51).

Suposiciones del compilador para tipos de dato integrales y de coma flotante

- Ejemplo de un posible problema:

```
short a, b, c;
a = 1 ;
b = 2 ;
c = a + b ; //compiler error
```

- Ejemplo de posibles soluciones:

- Declarar `c` como tipo `int` en la declaración original:

```
int c;
```

- Convertir el tipo del resultado de `(a+b)` en la línea de asignación:

```
c = (short) (a+b);
```

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

El compilador de tecnología Java realiza determinadas suposiciones cuando evalúa expresiones. Debe comprender estas suposiciones para realizar las conversiones de tipo adecuadas y otras adaptaciones.

Tipos de dato integrales y operaciones

La mayoría de las operaciones dan como resultado `int` o `long`:

- Los valores `byte`, `char` y `short` se amplían a `int` antes de la operación.
- Si alguno de los argumentos es del tipo `long`, el otro también se amplía a `long` y el resultado es del tipo `long`.

```
byte b1 = 1, b2 = 2, b3;
```

```
b3 = b1 + b2; // Error: result is an int but b3 is a byte
```

Ampliación a valores float

- Si una expresión contiene un valor `float`, la expresión entera se amplía a `float`. Todos los valores literales con coma flotante se ven como `double`.

En el siguiente ejemplo, se produce un error porque dos de los tres operandos (a y b) se amplían automáticamente de un tipo `short` a un tipo `int` antes de que se sumen:

```
short a, b, c;  
a = 1 ;  
b = 2 ;  
c = a + b ; //compiler error
```

En la última línea, los valores a y b se convierten a tipos `int` y los valores convertidos se suman para proporcionar un resultado `int`. A continuación, el operador de asignación (=) intenta asignar el resultado `int` a la variable `short` (c). Sin embargo, esta asignación no es válida y produce un error del compilador.

El código funciona si realiza lo siguiente:

- Declarar c como `int` en la declaración original:

```
int c;
```
- Convertir el tipo del resultado de (a+b) en la línea de asignación:

```
c = (short) (a+b);
```

Tipos de dato de coma flotante y asignación

- Ejemplo de un posible problema:

```
float float1 = 27.9; //compiler error
```

- Ejemplo de posibles soluciones:

- La F notifica al compilador que 27.9 es un valor float:

```
float float1 = 27.9F;
```

- 27.9 se convierte a un tipo float:

```
float float1 = (float) 27.9;
```



Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Al igual que los tipos integrales se definen por defecto en `int` en determinadas circunstancias, los valores asignados a tipos de coma flotante siempre se definen por defecto en un tipo `double`, a menos que indique específicamente que el valor es de tipo `float`.

Por ejemplo, la siguiente línea provoca un error del compilador. Puesto que se supone que 27.9 es un tipo `double`, se produce un error del compilador porque un valor de tipo `double` no puede encajar en una variable `float`.

```
float float1 = 27.9; //compiler error
```

Los dos siguientes funcionan correctamente:

- La F notifica al compilador que 27.9 es un valor float:

```
float float1 = 27.9F;
```

- 27.9 se convierte a un tipo float:

```
float float1 = (float) 27.9;
```

Ejemplo

```
public class Person {  
  
    public int ageYears = 32;  
  
    public void calculateAge() {  
  
        int ageDays = ageYears * 365;  
        long ageSeconds = ageYears * 365 * 24L * 60 * 60;  
  
        System.out.println("You are " + ageDays + " days old.");  
        System.out.println("You are " + ageSeconds + " seconds  
old.");  
  
    } // end of calculateAge method  
} // end of class
```

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

El ejemplo de código utiliza principios de esta sección para calcular la edad de una persona en días y segundos.

Prueba

¿Qué afirmación es verdadera?

- a. Hay ocho tipos primitivos incorporados en el lenguaje de programación Java.
- b. `byte`, `short`, `char` y `long` son los cuatro tipos de dato primitivos integrales del lenguaje de programación Java.
- c. Una variable de tipo `boolean` contiene `true`, `false` y `nil`.
- d. `long=10;` es un nombre de variable válido que cumple la sintaxis de declaración e inicialización de variables.



Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Respuesta: a

- a es correcta.
- b es incorrecta. Debe ser `byte`, `short`, `int` y `long`.
- c es incorrecta porque una variable de tipo `boolean` solo contiene `true` y `false`.
- d es incorrecta porque la palabra `long` es una palabra clave reservada.

Resumen

En esta lección, debe haber aprendido lo siguiente:

- Identificar los usos de las variables y definir la sintaxis de una variable
- Enumerar los ocho tipos de dato primitivos del lenguaje de programación Java
- Declarar, inicializar y utilizar variables y constantes según las instrucciones del lenguaje de programación Java y los estándares de codificación
- Modificar valores de variables mediante operadores
- Utilizar la ampliación y la conversión de tipo



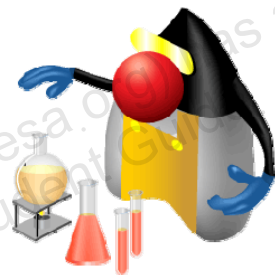
ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Visión general de la práctica 5-1: Declaración de variables de campo en una clase

En esta práctica, realizará las siguientes tareas:

- Creación de una clase que contenga varios campos
- Declaración de variables de campo y su inicialización
- Prueba de la clase mediante la ejecución un programa de prueba proporcionado



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Visión general de la práctica 5-2: Uso de operadores y conversión de tipo para evitar la pérdida de datos

En esta práctica, realizará las siguientes tareas:

- Uso de operadores para calcular la edad
- Uso de la conversión de tipo para evitar la pérdida de datos
- Creación de un programa de temperaturas para convertir Fahrenheit en Celsius

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

6

Trabajar con objetos

ORACLE®

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos

Al finalizar esta lección, debería estar capacitado para:

- Declarar, instanciar e inicializar variables de referencia de objetos
- Comparar cómo se almacenan las variables de referencia de objetos en relación con las variables primitivas
- Acceder a campos de objetos
- Llamar a métodos de objetos
- Crear un objeto String
- Manipular datos mediante la clase String y sus métodos
- Manipular datos mediante la clase StringBuilder y sus métodos
- Utilizar la documentación de la API de Java para explorar los métodos de una clase base



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Temas

- Declaración, instanciación e inicialización de objetos
- Trabajar con referencias de objetos
- Uso de la clase String
- Uso de la documentación de la API de Java
- Uso de la clase StringBuilder

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Trabajar con objetos: Introducción

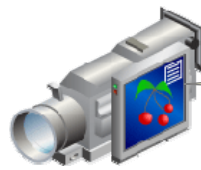
Se accede a los objetos a través de referencias.

- Los objetos son versiones instanciadas de su clase.
- Los objetos constan de atributos y operaciones:
 - En Java, son campos y métodos.

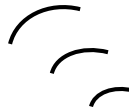
ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Acceso a objetos mediante una referencia



La cámara es como el objeto al que se accede a través de la referencia (control remoto).



El control remoto es como la referencia utilizada para acceder a la cámara (objeto).

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Para trabajar con un objeto, tiene que acceder a él a través de una referencia. Una buena analogía es el uso de un control remoto para manejar un dispositivo electrónico. Los botones del control remoto se pueden utilizar para modificar el funcionamiento del dispositivo (en este caso, una cámara). Por ejemplo, puede utilizar el control remoto para que la cámara se pare, reproduzca o grabe al interactuar con el control remoto.

Clase shirt

```
public class Shirt {
    public int shirtID = 0; // Default ID for the shirt
    public String description =
        "-description required-"; // default
    // The color codes are R=Red, B=Blue, G=Green, U=Unset
    public char colorCode = 'U';
    public double price = 0.0; // Default price all items
    // This method displays the details for an item
    public void display() {
        System.out.println("Item ID: " + shirtID);
        System.out.println("Item description:" +
            description);
        System.out.println("Color Code: " + colorCode);
        System.out.println("Item price: " + price);
    } // end of display method
} // end of class
```

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En este tema se trata el acceso a un objeto simple basado en la clase `Shirt` mostrada en la diapositiva. Tiene cuatro campos, `shirtID`, `description`, `colorCode` y `price`, y un método, `display()`. Tenga en cuenta que los métodos se suelen escribir de esta forma, con el nombre de método seguido por un par de paréntesis para indicar que es un método.

Puede observar que el método anteriormente denominado `displayInformation()` ahora se denomina solo `display()`. Aunque normalmente es mejor dar a los métodos nombres más descriptivos como `displayInformation()`, se utilizará `display()` en el resto del curso para que los ejemplos de código sean más compactos y fáciles de leer.

Temas

- Declaración, instanciación e inicialización de objetos
- **Trabajar con referencias de objetos**
- Uso de la clase String
- Uso de la documentación de la API de Java
- Uso de la clase StringBuilder

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Trabajar con variables de referencia de objetos

Declaración:

```
Classname identifier;
```

Instanciación:

```
new Classname ();
```

Este fragmento de código crea el objeto.

Asignación:

```
Object reference = new Classname ();
```

Identificador del paso de declaración

Operador de asignación

Para realizar la asignación a una referencia, la creación y la asignación deben estar en la misma sentencia.

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

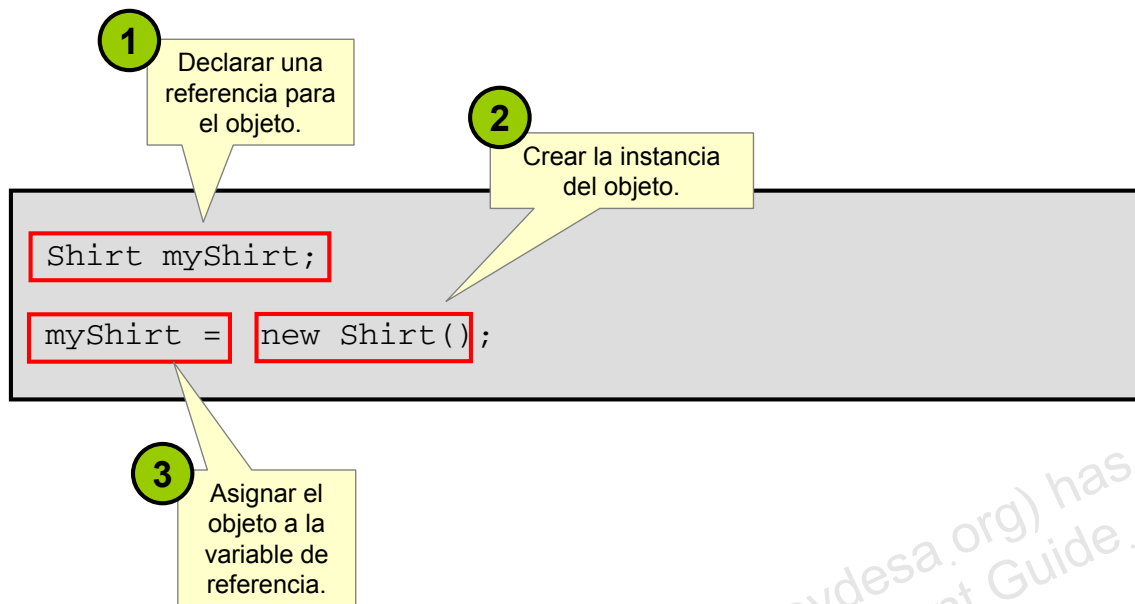
Existen tres pasos para obtener una referencia de objeto:

1. Declarar la referencia.
2. Instanciar el objeto.
3. Asignar el objeto a la referencia.

Tenga en cuenta que, como se indica en la diapositiva, el funcionamiento del operador de asignación (símbolo =) necesita que la referencia y el objeto recién creado estén en la misma sentencia. (Las sentencias acaban con el símbolo de punto y coma y no son lo mismo que líneas. El final de una línea no significa nada para el compilador Java; solo ayuda a que el código se pueda leer mejor).

El operador de asignación para asignar objetos a referencias es exactamente el mismo que el operador de asignación para asignar valores primitivos. No lo confunda con el símbolo == (igualdad). Aprenderá más tarde para qué se utiliza el símbolo de igualdad en Java.

Declaración e inicialización: Ejemplo

**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

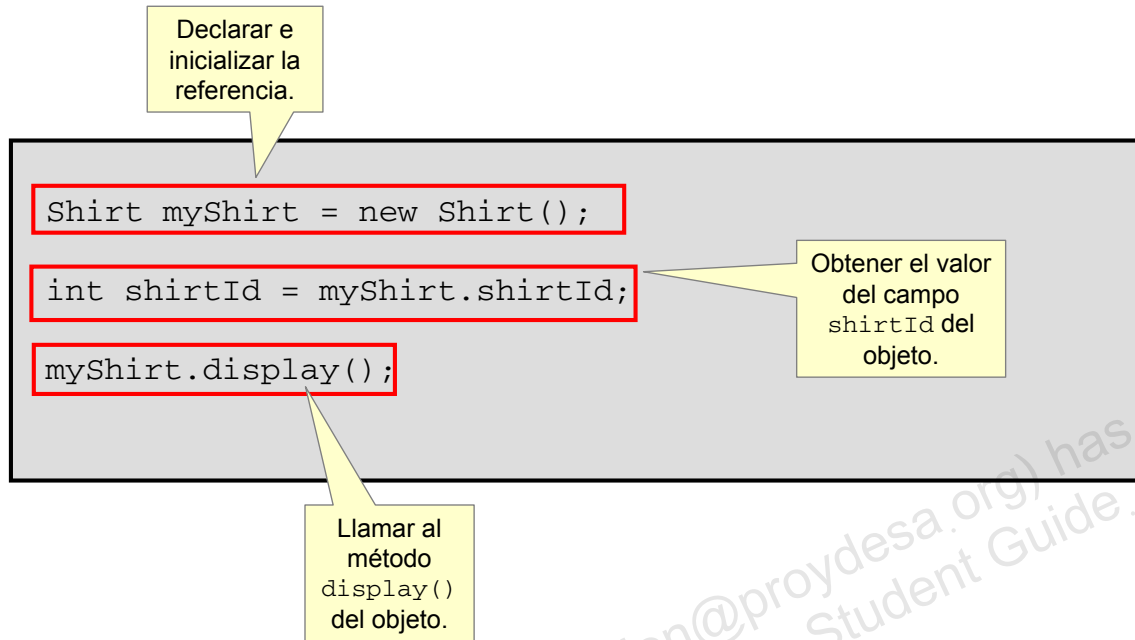
La declaración e inicialización de una variable de referencia es muy similar a la declaración e inicialización de una variable de tipo primitivo.

La principal diferencia es que debe crear una instancia del objeto (de una clase) para que la variable de referencia apunte a ella antes de inicializar la instancia del objeto.

Para declarar, instanciar e inicializar una variable de referencia de objeto:

1. Declare una referencia al objeto y especifique su identificador y el tipo de objeto al que apunta la referencia (la clase del objeto).
2. Cree la instancia del objeto mediante la palabra clave `new`.
3. Inicialice la variable de referencia de objeto mediante la asignación del objeto a dicha variable.

Trabajar con referencias de objetos

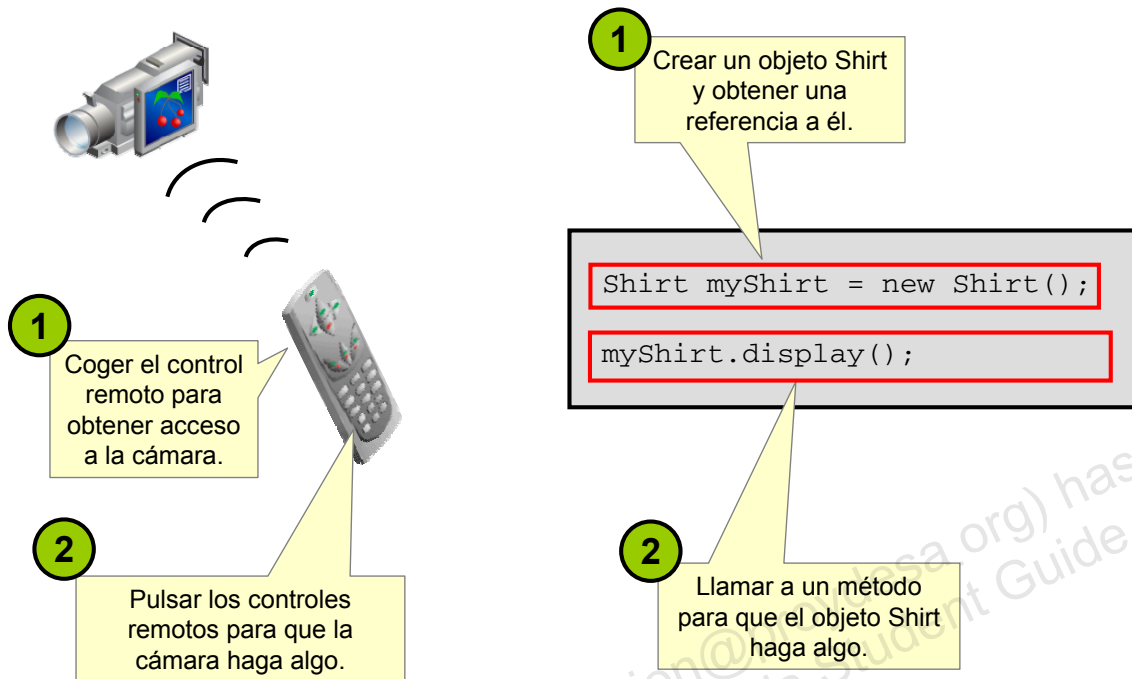
**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En el diagrama se ilustran algunos puntos importantes. Observe cómo la primera línea declara e inicializa la referencia de objeto en una sola línea (frente a dos líneas en la diapositiva anterior). Observe también el uso del operador de punto (.) con una referencia de objeto para manipular los valores o para llamar a los métodos de un objeto concreto. En el ejemplo de la diapositiva se utiliza la notación de puntos para acceder a un campo del objeto, en este caso mediante su asignación a una variable denominada `shirtId`.

La línea final de código del ejemplo muestra el uso de la notación de puntos para llamar a un método en el objeto.

Trabajar con referencias de objetos



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

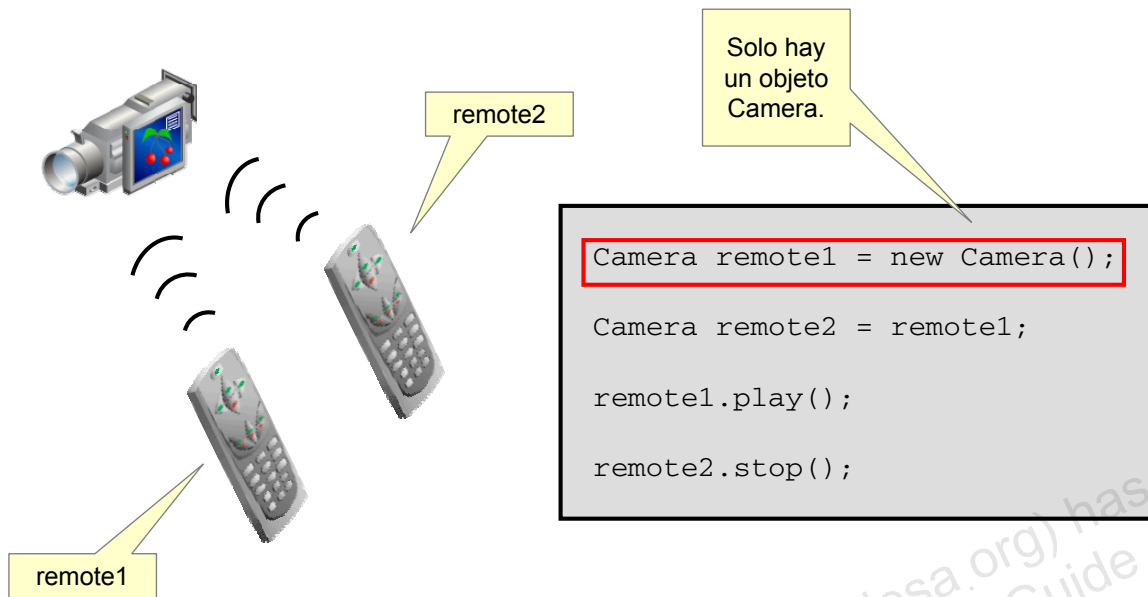
Volvamos a la analogía del uso de un control remoto para manejar un dispositivo electrónico. Para manejar un dispositivo electrónico con un control remoto, necesita:

1. Cogér el control remoto (y posiblemente encenderlo)
2. Pulsar un botón del control remoto para hacer algo en la cámara

Igualmente, para hacer algo con un objeto Java, necesita:

1. Obtener el “control remoto” (denominado referencia)
2. Pulsar los “botones” (denominados métodos)

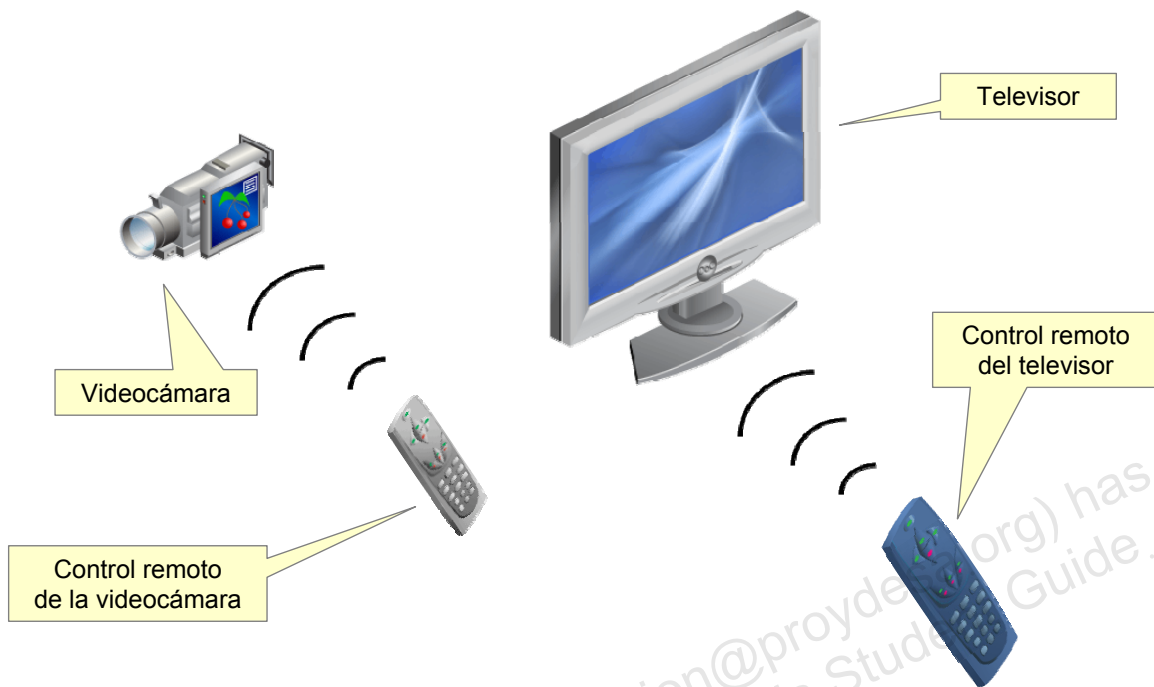
Trabajar con referencias de objetos

**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En el diagrama se muestra otro aspecto importante del funcionamiento de las referencias. En este ejemplo, se crea un objeto Camera y la referencia se asigna a una referencia de Camera, remote1. A continuación, esta referencia se asigna a otra referencia de Camera, remote2. Ahora ambas referencias están asociadas al mismo objeto Camera y los métodos llamados en cualquiera de las referencias afectarán al mismo objeto Camera.

Referencias a diferentes objetos

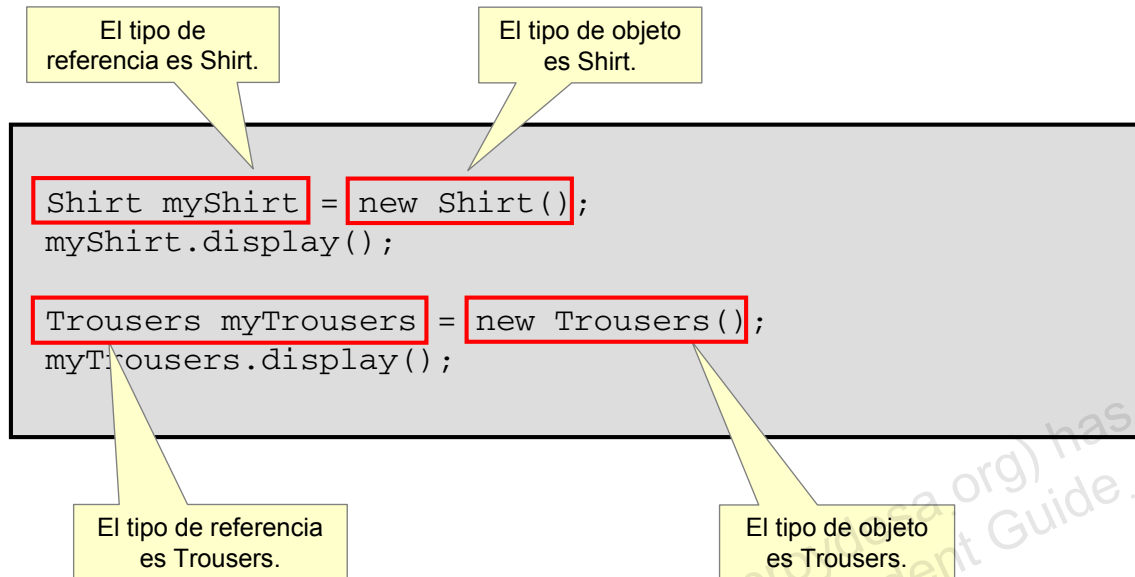
**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Para ampliar la analogía un poco más, para trabajar con otro objeto (por ejemplo, un televisor de pantalla plana), necesita un control remoto para dicho objeto. En el mundo de Java, necesita una referencia del tipo correcto para el objeto al que hace referencia.

Puede ignorar el hecho de que existen los controles remotos universales, aunque más adelante en el curso descubrirá que Java también tiene el concepto de referencias que no se limitan a un único tipo de objeto. Por el momento, digamos simplemente que una referencia del mismo tipo que un objeto es uno de los tipos de referencias que se pueden utilizar y que es un buen punto de partida para explorar el mundo de los objetos Java.

Referencias a diferentes tipos de objetos

**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

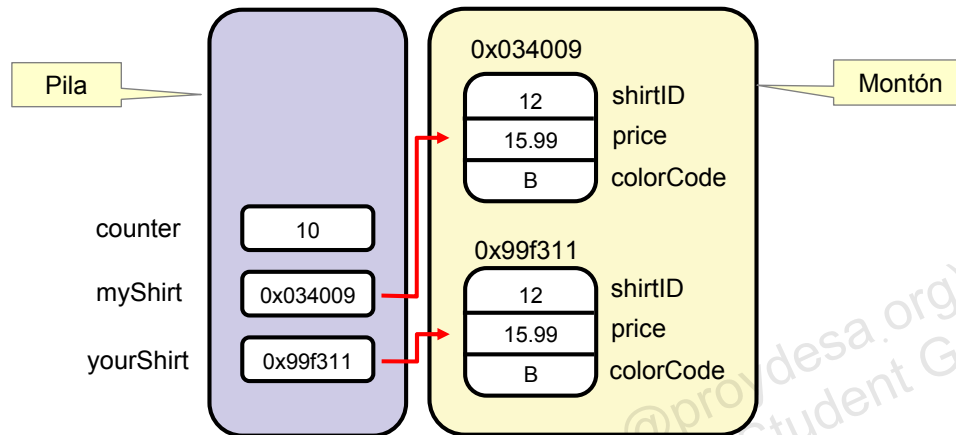
En el código de ejemplo de la diapositiva se muestran objetos a los que se accede con tipos de referencias coincidentes.

En el ejemplo, el tipo de referencia `Shirt` se utiliza para hacer referencia a un objeto `Shirt` y un tipo de referencia `Trousers` se utiliza para hacer referencia a un objeto `Trousers`.

Posteriormente, verá que el tipo de la referencia no tiene que ser idéntico al tipo del objeto, sino que tiene que ser compatible con él. Esta flexibilidad es un gran punto fuerte de Java y aprenderá más sobre ella en la lección titulada “Descripción de conceptos orientados a objetos avanzados”.

Referencias y objetos en memoria

```
int counter = 10;
Shirt myShirt = new Shirt();
Shirt yourShirt = new Shirt();
```



ORACLE

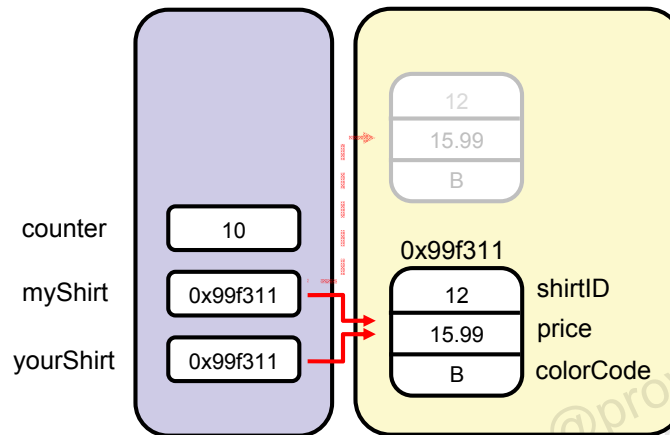
Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En este diagrama se muestra cómo las referencias apuntan a un objeto concreto en memoria. Tenga en cuenta que hay dos objetos en memoria, aunque ambos son de tipo **Shirt**. Tenga también en cuenta que hay dos referencias **Shirt** que apuntan a estos dos objetos **Shirt**.

En el diagrama también se muestran dos tipos de memoria que utiliza Java: la pila y el montón. La pila contiene variables locales, primitivas o tipos de referencia, mientras que el montón contiene objetos. Más adelante en este curso, aprenderá un poco más sobre las variables locales, pero por el momento es suficiente saber que las variables locales no son campos de un objeto.

Asignación de una referencia a otra

```
myShirt = yourShirt;
```



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En el diagrama se muestra qué sucede si la referencia `myShirt`, después de tener su propio objeto (en la diapositiva anterior), se asigna ahora a la referencia `yourShirt`. Cuando esto sucede, la referencia `myShirt` dejará su objeto actual y se reasignará al mismo objeto que tiene `yourShirt`. Como resultado, dos referencias, `myShirt` y `yourShirt`, apuntan ahora al mismo objeto. Se puede acceder a cualquier cambio en el objeto realizado con una referencia mediante la otra referencia, y viceversa.

Otro efecto de la asignación de la referencia `yourShirt` a la referencia `myShirt` es que si se el objeto anterior al que se hace referencia mediante `myShirt` no tiene ninguna otra referencia, ahora será inaccesible. En su debido momento, será basura recolectada, lo que significa que su memoria estará disponible para almacenar otros objetos.

Dos referencias, un objeto

Fragmento de código:

```
Shirt myShirt = new Shirt();  
Shirt yourShirt = new Shirt();  
  
myShirt = yourShirt;  
  
myShirt.colorCode = 'R';  
yourShirt.colorCode = 'G';  
  
System.out.println("Shirt color: " + myShirt.colorCode);
```

Salida del fragmento de código:

```
Shirt color: G
```

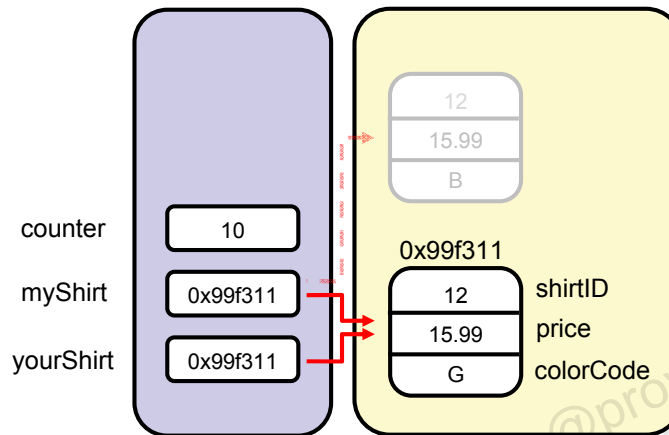
ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Este ejemplo ahora muestra qué sucede si utiliza una de las referencias para realizar un cambio u obtener un valor del objeto. Las referencias `yourShirt` y `myShirt` hacen referencia al mismo objeto, por lo que realizar un cambio u obtener un valor de campo con una es exactamente igual que hacerlo con la otra.

Asignación de una referencia a otra

```
myShirt.colorCode = 'R';
yourShirt.colorCode = 'G';
```


ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Puesto que `myShirt` y `yourShirt` ahora hacen referencia al mismo objeto después de que termine el código de la diapositiva, el campo `colorCode` del objeto será G. Y, por supuesto, obtendrá el mismo resultado si utiliza un código de los siguientes:

```
System.out.println(myShirt.colorCode);
System.out.println(yourShirt.colorCode);
```

Volviendo al ejemplo del control remoto del televisor, es lo mismo que si usted y un amigo hacen funcionar controles remotos con el mismo televisor.

Prueba

¿Cuál de las siguientes líneas de código instancia un objeto Boat y lo asigna a una referencia de objeto sailBoat?

- a. `Boat sailBoat = new Boat();`
- b. `Boat sailBoat;`
- c. `Boat = new Boat();`
- d. `Boat sailBoat = Boat();`

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Respuesta: a

Temas

- Declaración, instanciación e inicialización de objetos
- Trabajar con referencias de objetos
- **Uso de la clase String**
- Uso de la documentación de la API de Java
- Uso de la clase StringBuilder

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Clase String

La clase String soporta alguna sintaxis no estándar.

- Se puede instanciar un objeto String sin utilizar la palabra clave `new`; se prefiere esto:

```
String hisName = "Fred Smith";
```

- Se puede utilizar la palabra clave `new`, pero *no* se recomienda:

```
String herName = new String("Anne Smith");
```

- Un objeto String es inmutable; su valor no se puede cambiar.
- Un objeto String se puede utilizar con el símbolo del operador de concatenación de cadenas (+) para la concatenación.



Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

- La clase String es una de las muchas clases incluidas en las bibliotecas de clases Java. La clase String proporciona la capacidad de almacenar una secuencia de caracteres. Utilizará la clase String frecuentemente en sus programas. Por lo tanto, es importante comprender algunas de las características especiales de las cadenas en el lenguaje de programación Java.
- Al crear un objeto String con la palabra clave `new`, se crean dos objetos String en memoria, mientras que al crear un objeto String con un literal string, se crea solo un objeto; por lo tanto, esta última práctica es más eficaz en cuanto a la memoria. Para evitar la duplicación innecesaria de objetos String en memoria, cree los objetos String sin la palabra clave `new`.

Concatenación de cadenas

Cuando utiliza un literal de cadena en el código Java, se instancia y se convierte en una referencia String.

- Concatenar cadenas:

```
String name1 = "Fred"  
theirNames = name1 + " and " +  
               "Anne Smith";
```
- La concatenación crea una nueva cadena y la referencia String `theirNames` apunta ahora a esta nueva cadena.

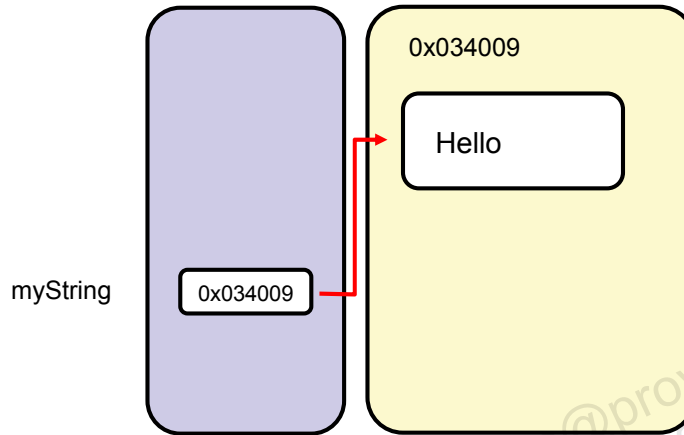


Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Los objetos String soportan el uso de un operador de concatenación especial (+) para la concatenación de dos o más cadenas. Puesto que una cadena de literales devuelve una referencia String, los literales de cadena y las referencias String se pueden mezclar en una expresión que concatena un número de cadenas, como se muestra en la diapositiva.

Concatenación de cadenas

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

**ORACLE**

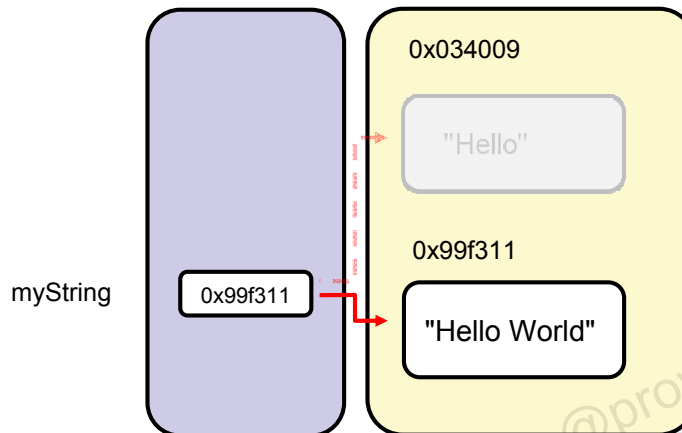
Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Puesto que String es inmutable, la concatenación de dos cadenas necesita la creación de una nueva cadena.

En el diagrama se muestra un objeto String que contiene la cadena "Hello".

Concatenación de cadenas

```
String myString = "Hello";  
myString = myString.concat(" World");
```

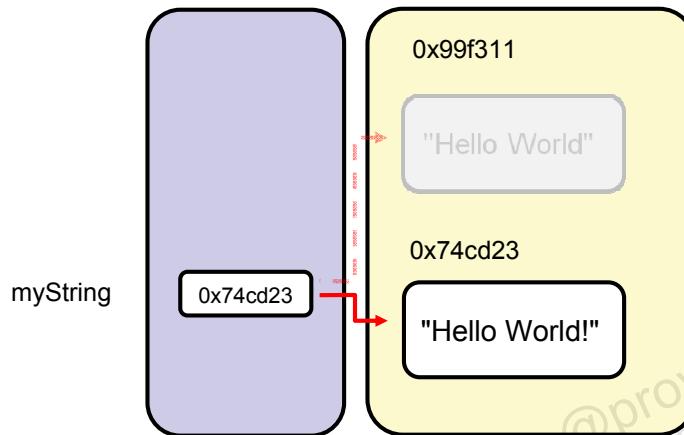
**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Aquí está a cadena "World" concatenada a la cadena original. Aquí se utiliza el método `concat()`, pero tanto si utiliza éste método como el operador de concatenación (+), se crea un nuevo objeto String y se devuelve una nueva referencia String que apunta a este nuevo objeto. En el diagrama, esto se muestra con el hecho de que la referencia String `myString` ya no es 0x034009 y porque ya no se hace referencia a ese objeto, ahora es inaccesible y será basura recopilada.

Concatenación de cadenas

```
String myString = "Hello";  
myString = myString.concat(" World");  
myString = myString + "!"
```

**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Finalmente, al concatenar otra cadena, esta vez con el operador de concatenación, vuelve a suceder lo mismo. Se crea un nuevo objeto y la referencia de este objeto se asigna a `myString`.

Llamadas al método String con valores de retorno primitivos

Una llamada a método puede devolver un único valor de cualquier tipo.

- Ejemplo de un método de tipo primitivo `int`:

```
String hello = "Hello World";  
int stringLength = hello.length();
```

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Como la mayoría de clases, la clase `String` tiene una serie de métodos útiles. Casi todos estos métodos hacen su trabajo útil con la devolución de un único valor (Java solo permite un único retorno de un método). El tipo de retorno (esencialmente el tipo del método) puede ser primitivo o una referencia a un objeto.

Para poder utilizar el valor de retorno en el código, normalmente utilizará el operador de asignación para asignar el valor (o la referencia) a un tipo que haya declarado para este objetivo.

El ejemplo de la diapositiva muestra el uso de la referencia `hello` para llamar al método `length()`. Puesto que el objeto al que hace referencia esta referencia es la cadena `Hello World`, esta llamada a método devolverá el valor `11` y lo colocará en la variable `stringLength`. `int` es el tipo de la llamada a método `length()`.

Llamadas al método String con valores de retorno de objeto

Llamadas a método que devuelven objetos:

```
String greet = " HOW ".trim();
String lc = greet + "DY".toLowerCase();
```

O bien

```
String lc = (greet + "DY").toLowerCase();
```

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En este ejemplo se muestran varias llamadas a método que devuelven referencias de objetos.

En primer lugar, se instancia el objeto de cadena " HOW " y se llama a `trim` en él. A medida que un literal de cadena devuelve una referencia de objeto, es exactamente lo mismo que llamar al método `trim()` en la referencia. Observe que la cadena " HOW " tiene dos espacios a ambos lados de la palabra. La cadena devuelta será solo de tres caracteres porque estos espacios se eliminarán. Se hará referencia a esta cadena nueva mediante "greet".

En el siguiente ejemplo se muestra una llamada a método que no se asigna a un tipo, sino que simplemente se utiliza en una expresión. Se llama a `toLowerCase()` en la cadena "DY", que devuelve "dy". `lc` ahora hace referencia a un objeto que contiene "HOWdy".

Finalmente, observe cómo una versión alternativa con paréntesis garantiza que las dos cadenas estén concatenadas (creando una nueva cadena) antes de que se llame a `toLowerCase()`. `lc` ahora hace referencia a un objeto que contiene "howdy".

Llamadas a métodos que necesitan argumentos

Las llamadas a métodos pueden necesitar transferir uno o más argumentos:

- Transferir un primitivo

```
String theString = "Hello World";  
String partString = theString.substring(6);
```

- Transferir un objeto

```
boolean endWorld =  
    "Hello World".endsWith("World");
```



Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Algunas llamadas a métodos necesitan transferir un argumento al método.

Por ejemplo, el método `substring()` que se muestra en el ejemplo necesita un índice (`int`) para indicar dónde dividir la cadena. Devuelve una nueva cadena que consta de la parte restante de la cadena que empieza en “w”, por lo que en este caso devuelve “World”. (La subcadena se indexa a partir de 0, empieza con el carácter del índice especificado y se amplía hasta el final de esta cadena. “w” está en el índice 6).

El método `endsWith()` necesita que se transfiera una referencia `String` como argumento. Devuelve un valor `boolean` porque simplemente determina si la cadena finaliza con la secuencia de caracteres transferidos. En este caso lo hace, por lo que se devolverá `true`.

Temas

- Declaración, instanciación e inicialización de objetos
- Trabajar con referencias de objetos
- Uso de la clase String
- **Uso de la documentación de la API de Java**
- Uso de la clase StringBuilder

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Documentación de la API de Java

Consta de un juego de páginas web.

- Muestra todas las clases de la API
 - Descripciones de la función de la clase
 - Lista de constructores, métodos y campos de la clase
- Gran cantidad de hiperenlaces para mostrar las interconexiones entre las clases y facilitar la búsqueda
- Disponible en el sitio web de Oracle en:
<http://download.oracle.com/javase/7/docs/api/index.html>



Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Todos los JDK de tecnología Java contienen una serie de clases escritas previamente que puede utilizar en los programas. Estas bibliotecas de clases de tecnología Java se incluyen en la documentación de la API de Java para la versión del JDK que utiliza. La especificación de biblioteca de clases es una serie de páginas web HTML que puede cargar en el explorador web.

Una especificación de biblioteca de clases Java es un documento muy detallado que describe las clases de la API. Cada API incluye documentación que describe el uso de las clases, así como sus campos y métodos. Cuando busca una forma de realizar un determinado juego de tareas, esta documentación es la mejor fuente de información sobre las clases desarrolladas previamente en las bibliotecas de clases Java.

Documentación de la plataforma Java SE 7

Aquí puede seleccionar All Classes o un paquete concreto.

En este panel se muestran detalles sobre la clase seleccionada.

Según lo que seleccione, aquí se muestran las clases de un paquete concreto o todas las clases.

The screenshot shows the Java Platform Standard Ed. 7 documentation interface. The top-left panel displays a list of packages and classes. The top-right panel shows the details for the 'Class String', including its inheritance, implemented interfaces, and the class declaration. The bottom panel shows the class declaration and a description of the String class.

ORACLE®

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

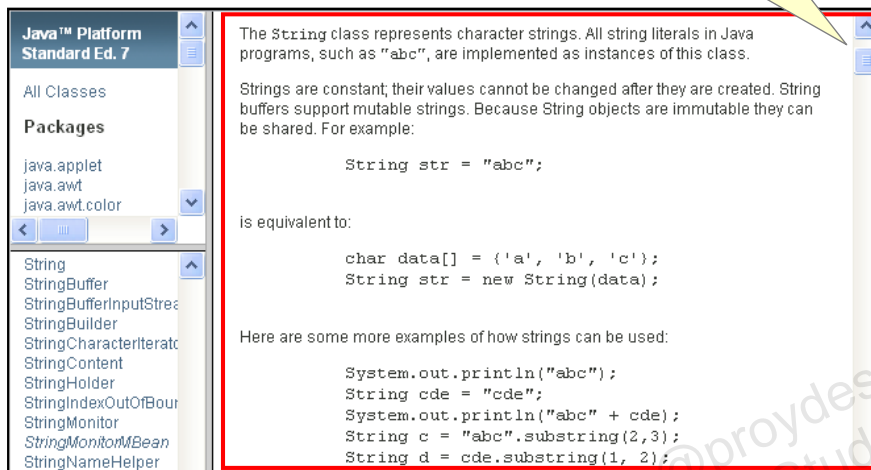
En la captura de pantalla de la diapositiva, puede ver los tres paneles principales de la página web.

El panel superior derecho permite seleccionar un paquete. Las clases Java se organizan en paquetes, pero si no sabe el paquete de una clase concreta, puede seleccionar All Classes.

El panel inferior izquierdo ofrece la lista de clases de un paquete, o bien todas las clases si ha seleccionado la opción correspondiente. En este panel, se ha seleccionado la clase String, rellenando el panel principal de la derecha con los detalles de la clase String.

Documentación de la plataforma Java SE 7

Al desplazarse hacia abajo, se muestra más información de la clase String.



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

El panel principal de la derecha contiene mucha información sobre la clase, por lo que tiene que desplazarse hacia abajo para acceder a la información que necesita.

Plataforma Java SE 7: Resumen del método

The screenshot shows the 'Method Summary' for the `String` class. It lists several methods with their return types, names, and parameters. Three callouts highlight specific details:

- Tipo del método (tipo que se devuelve).** Points to the `char` return type of the `charAt` method.
- Tipo del parámetro que se debe transferir al método.** Points to the `int index` parameter of the `charAt` method.
- Nombre del método.** Points to the `charAt` method name.

Modifier and Type	Method and Description
<code>char</code>	<code>charAt(int index)</code> Returns the char value at the specified index.
<code>int</code>	<code>codePointAt(int index)</code> Returns the character (Unicode code point) at the specified index.
<code>int</code>	<code>codePointBefore(int index)</code> Returns the character (Unicode code point) before the specified index.
<code>int</code>	<code>indexOf(int ch, int fromIndex, int endIndex)</code> Returns the index of the first occurrence of the specified character in the specified text.
<code>int</code>	<code>indexOf(String str)</code> Compares two strings lexicographically.
<code>int</code>	<code>compareToIgnoreCase(String str)</code> Compares two strings lexicographically, ignoring case differences.
<code>String</code>	<code>concat(String str)</code> Concatenates the specified string to the end of this string.

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Si sigue desplazándose por los detalles de la clase `String`, llegará a la lista de métodos (aquí solo se muestra un pequeño subjuego de esta lista).

Esta lista maestra de métodos proporciona los detalles básicos del método. En este caso, puede ver que el tipo de método se llama `charAt()`, su tipo es `char` y necesita que se transfiera un índice (de tipo `int`). También hay una breve descripción que indica que este método devuelve el valor `char` en un índice concreto de la cadena.

Plataforma Java SE 7: Detalles del método

Haga clic aquí para obtener la descripción detallada del método.

```
int indexOf(String str)
Returns the index within this string of the first occurrence of the
specified substring.
int indexOf(String str, int fromIndex)
Returns the index within this string of the first occurrence of the
specified substring, starting at the specified index.
```

Descripción detallada del método `indexOf()`.

indexOf

```
public int indexOf(String str)
```

Returns the index within this string of the first occurrence of the specified substring.

The returned index is the smallest value *k* for which:

```
this.startsWith(str, k)
```

If no such value of *k* exists, then -1 is returned.

Parameters:

str - the substring to search for.

Returns:

the index of the first occurrence of the specified substring, or -1 if there is no such occurrence.

Se muestran más detalles sobre los parámetros y el valor de retorno en la lista de método.

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Para cualquiera de los métodos, se hiperenlazan el nombre de método y los tipos de parámetros para que pueda obtener más información. En este ejemplo, se muestra la descripción detallada de uno de los métodos `indexOf()` de `String`.

Métodos `System.out`

Para encontrar todos los detalles de `System.out.println()`, considere lo siguiente:

- `System` es una clase (en `java.lang`).
- `out` es un campo de `System`.
- `out` es un tipo de referencia que permite llamar a `println()` en el tipo de objeto al que hace referencia.

Para buscar la documentación:

1. Vaya a la clase `System` y busque el tipo del campo `out`.
2. Vaya a la documentación de dicho campo.
3. Revise los métodos disponibles.



Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En las siguientes diapositivas se muestra cómo puede utilizar la documentación de la API de Java para obtener más información sobre `System.out.println()`. Como verá, esto es poco común, porque la clase cuyos métodos necesita investigar no es `System`. En su lugar, es la clase que es el tipo del campo `out` del objeto `System`.

Documentación sobre `System.out.println()`

Class System
 java.lang.Object
 java.lang.System
 public final class System
 extends Object

Field Summary

Modifier and Type	Field and Description
static PrintStream	err The "standard" error output stream.
static InputStream	in The "standard" input stream.
static PrintStream	out The "standard" output stream.

Method Summary

void	print(Object obj) Prints an object.
void	print(String s) Prints a string.
PrintStream	printf(Locale l, String format, Object... args) A convenience method to write a formatted string to this output.
void	println(double x) Prints a double and then terminate the line.
void	println(float x) Prints a float and then terminate the line.
void	println(int x) Prints an integer and then terminate the line.
void	println(long x) Prints a long and then terminate the line.
void	println(Object x) Prints an Object and then terminate the line.
void	println(String x) Prints a String and then terminate the line.

Algunos de los métodos de PrintStream

El campo out de System es de tipo PrintStream.

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En el diagrama se muestra Field Summary para la clase `System`. Aquí, puede ver que efectivamente hay un campo llamado `out` y que es de tipo `PrintStream`. Al hacer clic en `PrintStream`, ahora puede ver los detalles de esa clase y, si se desplaza hacia abajo hasta Method Summary, encontrará (entre muchos otros métodos) el método `print()` y el método `println()`.

Uso de los métodos `print()` y `println()`

- **Método `println()`:**
`System.out.println(data_to_print);`
- **Ejemplo:**
`System.out.print("Carpe diem ");`
`System.out.println("Seize the day");`
- **Este método muestra lo siguiente:**
Carpe diem Seize the day

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Tenga en cuenta que `System.out` es de tipo `PrintStream`, pero `PrintStream` no es solo para imprimir en la consola. Es solo que el valor por defecto para este campo es una referencia a un objeto `PrintStream` que muestra una salida en la consola, pero puede cambiar la referencia `out` si desea que la salida vaya a otro lugar.

La diferencia entre el método `print()` y el método `println()` es que `print()` no crea una nueva línea después de imprimir `String`, mientras que `println()` sí lo hace. Por consiguiente, en el ejemplo de la diapositiva, "Seize the day" aparece en la misma línea que "Carpe diem".

Temas

- Declaración, instanciación e inicialización de objetos
- Trabajar con referencias de objetos
- Uso de la clase String
- Uso de la documentación de la API de Java
- **Uso de la clase StringBuilder**

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Clase StringBuilder

StringBuilder proporciona una alternativa variable a String.

StringBuilder:

- Es una clase normal. Utilice `new` para instanciarla.
- Tiene un amplio juego de métodos para agregar, insertar y suprimir.
- Tiene muchos métodos para devolver una referencia al objeto actual. No hay ningún costo de instanciación.
- Se puede crear con la capacidad inicial que mejor se adapte a las necesidades.

String sigue siendo necesaria porque:

- Su uso puede ser más seguro que un objeto inmutable.
- Una clase de la API puede necesitar una cadena.
- Tiene muchos más métodos no disponibles en StringBuilder.

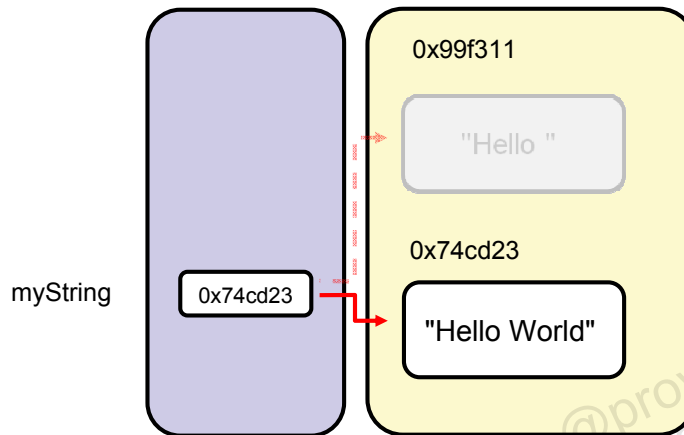
ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

A diferencia de String, no hay ningún método abreviado para instanciar la clase StringBuilder. Simplemente se instancia como cualquier otro objeto mediante el uso de la palabra clave `new`. StringBuilder no es una sustitución completa de String, pero es más adecuada si es probable que se realicen muchas modificaciones en la cadena representada por el tipo de dato.

Ventajas de StringBuilder sobre String para la concatenación (o adición)

```
String myString = "Hello";  
myString = myString.concat(" World");
```

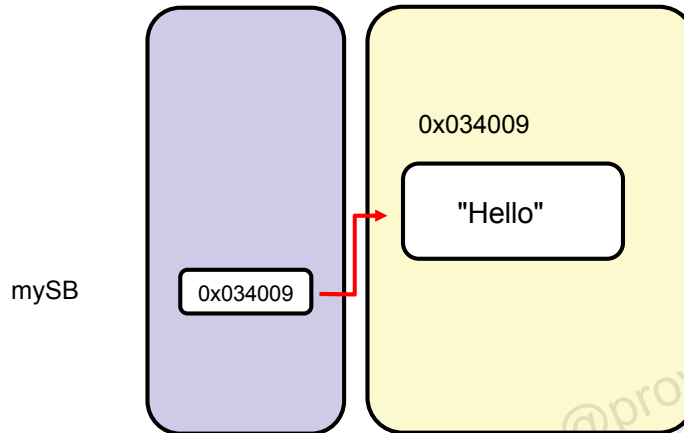
**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En esta diapositiva se ofrece un recordatorio de lo que sucede cuando las cadenas "Hello" y "World" se concatenan. Se crea un nuevo objeto String y la referencia de ese objeto se asigna a myString.

StringBuilder: Declaración e instanciación

```
StringBuilder mySB = new StringBuilder("Hello");
```

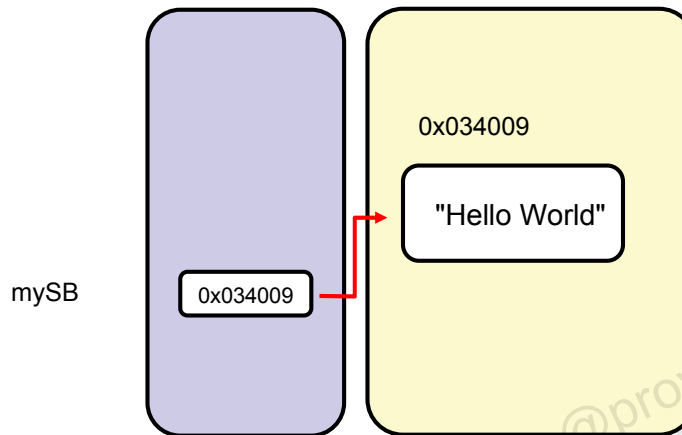
**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En este diagrama se muestra el inicio de una secuencia que implica un objeto `StringBuilder`. Se instancia un nuevo objeto `StringBuilder`, relleno con la cadena "Hello" y la referencia de este nuevo objeto se asigna a `mySB`.

Adición de StringBuilder

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



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Para agregar la cadena "World", todo lo que tiene que hacer es llamar al método `append()` y transferir "World". Tenga en cuenta que no es necesaria ninguna asignación porque ya hay una referencia al objeto `StringBuilder` y este objeto `StringBuilder` ahora contiene una representación de las cadenas combinadas "Hello World".

Incluso si asignó el tipo de retorno del método `append()` (que es `StringBuilder`), seguirá sin haber ningún costo de creación de objeto; el método `append()` modifica el objeto actual y devuelve la referencia a ese objeto, el que ya se incluye en `mySB`. (Esto puede resultar útil para saber si se utiliza la llamada a método completa como tipo).

Prueba

¿Cuáles de las siguientes afirmaciones son ciertas?
(Seleccione todas las respuestas posibles).

- a. El operador de punto (.) crea una nueva instancia de objeto.
- b. La clase String proporciona la capacidad de almacenar una secuencia de caracteres.
- c. La especificación de la API de Java contiene documentación para todas las clases de un producto de tecnología Java.
- d. Los objetos String no se pueden modificar.

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Respuesta: b, c, d

Resumen

Se accede a los objetos a través de referencias:

- Los objetos son versiones instanciadas de su clase.
- Los objetos constan de atributos y operaciones:
 - En Java, son campos y métodos.
- Para acceder a los campos y métodos de un objeto, obtenga una variable de referencia al objeto:
 - El mismo objeto puede tener más de una referencia.
- Una referencia de objeto existente se puede reasignar a una nueva variable de referencia.
- La palabra clave `new` instancia un nuevo objeto y devuelve una referencia.



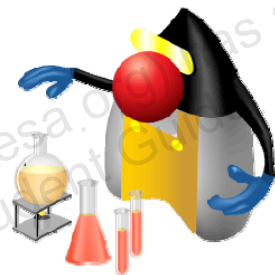
ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Visión general de la práctica 6-1: Creación y manipulación de objetos Java

En esta práctica, creará instancias de una clase y manipulará estas instancias de varias formas. En esta práctica, podrá:

- Crear e inicializar instancias de objeto
- Manipular referencias de objetos

**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Visión general de la práctica 6-2: Uso de la clase StringBuilder

En esta práctica, creará, inicializará y manipulará objetos StringBuilder.

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Visión general de la práctica 6-3: Examen de la especificación de la API de Java

En esta práctica, examinará la especificación de la API de Java para familiarizarse con la documentación y con la búsqueda de clases y métodos.

No se espera que comprenda todo lo que vea.

Sin embargo, a medida que avance en este curso, comprenderá cada vez más la documentación de la API de Java.

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Fundacion Proydesa (fundacion@proydesa.org) has a
non-transferable license to use this Student Guide.

Uso de operadores y construcciones de decisión

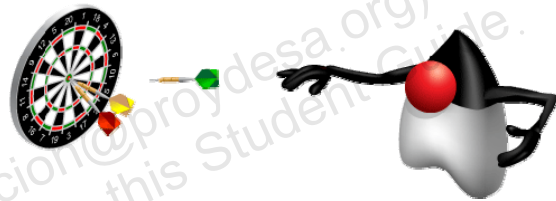
ORACLE®

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Objetivos

Al finalizar esta lección, debería estar capacitado para:

- Utilizar un operador relacional
- Probar la igualdad entre cadenas
- Utilizar un operador condicional
- Crear construcciones `if` e `if/else`
- Anidar una sentencia `if`
- Encadenar una sentencia `if/else`
- Utilizar una sentencia `switch`



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Importancia

- Cuando tiene que tomar una decisión para la que existan varios caminos diferentes, ¿cómo selecciona en última instancia un camino en lugar de los otros?
- Por ejemplo, ¿en qué cosas piensa cuando va a comprar un artículo?

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En nuestra vida diaria, tenemos que tomar muchas decisiones y a menudo utilizamos la palabra “si” con alguna condición cuando tomamos esas decisiones. Por ejemplo, “si la casa es azul, me daré una vuelta por ella”. O bien, “si el coche es deportivo y seguro, lo comparé”. Pensamos en estos tipos de decisiones de forma inconsciente cada día.

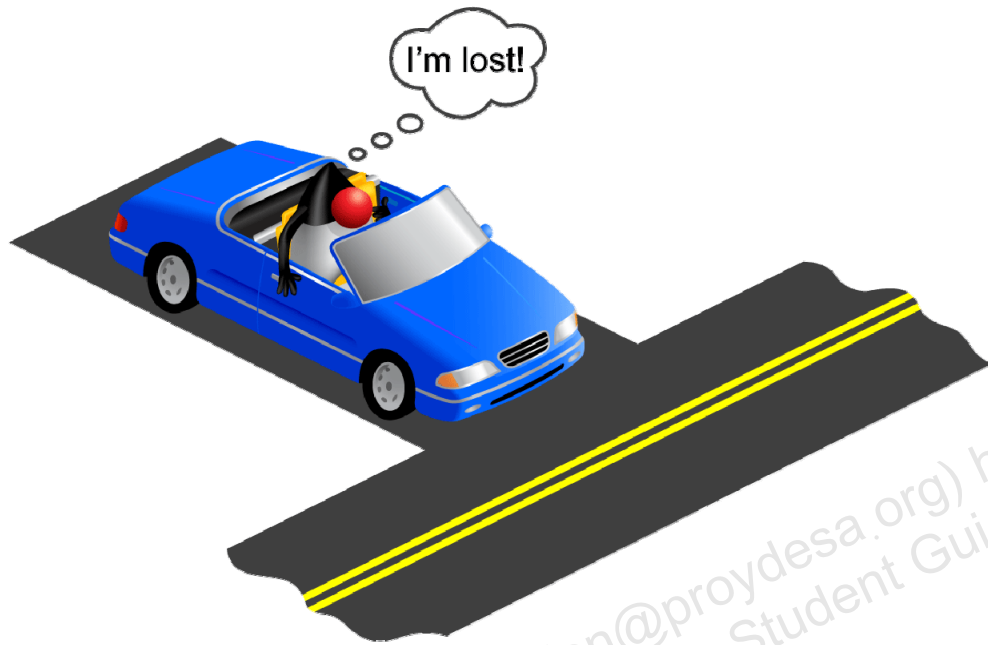
Temas

- **Uso de operadores relacionales y condicionales**
- Creación de construcciones `if` e `if/else`
- Encadenamiento de una sentencia `if/else`
- Uso de una sentencia `switch`

ORACLE®

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de operadores relacionales y condicionales

**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Una de las tareas que a menudo realizan los programas es evaluar una condición y, según el resultado, ejecutar distintos bloques o ramas de código. Por ejemplo, el programa puede comprobar si el valor de una variable es igual al valor de otra y, si es así, hacer algo. En la imagen se ilustra el tipo de decisión que las personas toman todos los días. Además de los operadores aritméticos, como el signo más (+) y el aumento (++), el lenguaje de programación Java proporciona varios operadores relacionales, incluidos < y > para “menor que” y “mayor que”, respectivamente, y && para “AND”. Estos operadores se utilizan cuando se desea que el programa ejecute diferentes bloques o ramas de código según las distintas condiciones, como la comprobación de si el valor de dos variables es el mismo.

Nota: cada uno de estos operadores se utiliza en el contexto de una construcción de decisión, como una construcción `if` o `if/else`, que se presentarán más adelante.

Ejemplo de ascensor

```
public class Elevator {

    public boolean doorOpen=false; // Doors are closed by default
    public int currentFloor = 1; // All elevators start on first floor
    public final int TOP_FLOOR = 10;
    public final int MIN_FLOORS = 1;

    public void openDoor() {
        System.out.println("Opening door.");
        doorOpen = true;
        System.out.println("Door is open.");
    }

    public void closeDoor() {
        System.out.println("Closing door.");
        doorOpen = false;
        System.out.println("Door is closed.");
    }

    ...
}
```

Abrir la puerta.

Cerrar la puerta.



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Un ascensor tiene muchas funciones. Empecemos por un ascensor que solo tiene la siguiente funcionalidad. (Esta funcionalidad se mejorará a medida que veamos más ejemplos en lecciones posteriores).

Las funciones del ascensor en esta lección son:

- Las puertas del ascensor se pueden abrir.
- Las puertas del ascensor se pueden cerrar.
- El ascensor puede subir una planta.
- El ascensor puede bajar una planta.

Verá diferentes variaciones de la clase `Elevator` en esta lección y en posteriores, incluidas distintas variaciones que ilustran el uso de construcciones de decisión. El código completo del ejemplo del ascensor para esta lección es el siguiente:

```
public class Elevator {

    public boolean doorOpen=false; // Default setting
    public int currentFloor = 1; // Default starting point
    public final int TOP_FLOOR = 10;
    public final int MIN_FLOORS = 1;

    public void openDoor() {
        System.out.println("Opening door.");
        doorOpen = true;
        System.out.println("Door is open.");
    }

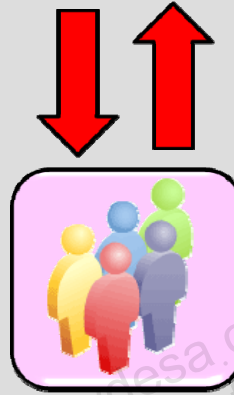
    public void closeDoor() {
        System.out.println("Closing door.");
        doorOpen = false;
        System.out.println("Door is closed.");
    }

    public void goUp() {
        System.out.println("Going up one floor.");
        currentFloor++;
        System.out.println("Floor: " + currentFloor);
    }

    public void goDown() {
        System.out.println("Going down one floor.");
        currentFloor--;
        System.out.println("Floor: " + currentFloor);
    }
}
```

Archivo ElevatorTest.java

```
public class ElevatorTest {  
    public static void main(String args[]) {  
  
        Elevator myElevator = new Elevator();  
  
        myElevator.openDoor();  
        myElevator.closeDoor();  
        myElevator.goDown();  
        myElevator.goUp();  
        myElevator.goUp();  
        myElevator.goUp();  
        myElevator.openDoor();  
        myElevator.closeDoor();  
        myElevator.goDown();  
        myElevator.openDoor();  
        myElevator.goDown();  
        myElevator.openDoor();  
    }  
}
```

**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Una clase de prueba similar a la del ejemplo realiza algunas pruebas de Elevator.

Operadores relacionales

Condición	Operador	Ejemplo
Es igual a	==	<code>int i=1;</code> <code>(i == 1)</code>
Es distinto de	!=	<code>int i=2;</code> <code>(i != 1)</code>
Es menor que	<	<code>int i=0;</code> <code>(i < 1)</code>
Es menor o igual que	<=	<code>int i=1;</code> <code>(i <= 1)</code>
Es mayor que	>	<code>int i=2;</code> <code>(i > 1)</code>
Es mayor o igual que	>=	<code>int i=1;</code> <code>(i >= 1)</code>

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Los operadores relacionales comparan dos valores para determinar su relación. En la tabla se muestran las diferentes condiciones que puede probar mediante operadores relacionales. El resultado de todos los operadores relacionales es un valor booleano. Los valores booleanos pueden ser true o false. Por ejemplo, todos los ejemplos de la tabla producen un resultado booleano true.

Nota: el signo igual (=) se utiliza para realizar una asignación.

Prueba de la igualdad entre cadenas

Ejemplo:

```
public class Employees {

    public String name1 = "Fred Smith";
    public String name2 = "Joseph Smith";

    public void areNamesEqual() {

        if (name1.equals(name2)) {
            System.out.println("Same name.");
        }
        else {
            System.out.println("Different name.");
        }
    }
}
```



ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Si utiliza el operador `==` para comparar referencias de objetos con objetos `String`, el operador prueba si las direcciones de las referencias de objetos `String` de la memoria son iguales, no su contenido.

Análisis: ¿Son iguales todas las referencias de objetos `String` siguientes?

```
String helloString1 = ("hello");
String helloString2 = "hello";
String helloString3 = new String("hello");
```

Si desea probar la igualdad entre las cadenas de caracteres (como si el nombre "Fred Smith" es igual a "Joseph Smith"), utilice el método `equals` de la clase `String`. La clase del ejemplo contiene dos nombres de empleados y un método para comparar los nombres.

Operadores condicionales comunes

Operación	Operador	Ejemplo
Si una condición AND otra condición	&&	<pre>int i = 2; int j = 8; ((i < 1) && (j > 6))</pre>
Si una condición OR otra condición		<pre>int i = 2; int j = 8; ((i < 1) (j > 10))</pre>
NOT	!	<pre>int i = 2; (!(i < 3))</pre>

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

También tendrá que poder tomar una única decisión según más de una condición. En dichas circunstancias, puede utilizar operadores condicionales para evaluar condiciones complejas como un todo. En la tabla de la diapositiva se muestran los operadores condicionales comunes en el lenguaje de programación Java. Por ejemplo, todos los ejemplos de la tabla producen un resultado booleano false.

Análisis: ¿Qué operadores relacionales y condicionales se expresan en el siguiente párrafo?

Si el juguete es rojo, lo compraré. Sin embargo, si el juguete es amarillo y cuesta menos que un artículo rojo, también lo compraré. Si el juguete es amarillo y cuesta lo mismo o más que otro artículo rojo, no lo compraré. Finalmente, si el juguete es verde, no lo compraré.

Operador condicional ternario

Operación	Operador	Ejemplo
Si <code>someCondition</code> es <code>true</code> , asigne el valor de <code>value1</code> al resultado. En caso contrario, asigne el valor de <code>value2</code> al resultado.	<code>?:</code>	<code>someCondition ? value1 : value2</code>

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

El operador ternario es un operador condicional que toma tres operandos. Necesita una sintaxis más breve que una sentencia `if/else`. Utilice el operador `?:` en lugar de una sentencia `if/else` si hace que el código sea más legible; por ejemplo, cuando las expresiones son compactas y sin efectos secundarios (como asignaciones). El primer operando es una expresión booleana.

Aprenderá sobre las sentencias `if/else` en la siguiente sección.

Temas

- Uso de operadores relacionales y condicionales
- **Creación de construcciones `if` e `if/else`**
- Encadenamiento de una sentencia `if/else`
- Uso de una sentencia `switch`

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Creación de construcciones `if` e `if/else`

Una sentencia `if`, o una construcción `if`, ejecuta un bloque de código si una expresión es *true*.

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Construcción `if`

- **Sintaxis:**

```
if (boolean_expression) {
    code_block;
} // end of if construct
// program continues here
```

- **Ejemplo de posible salida:**

```
Opening door.
Door is open.
Closing door.
Door is closed.
Going down one floor.
Floor: 0 ← Se trata de un error en la lógica.
Going up one floor.
Floor: 1
Going up one floor.
Floor: 2
...
```

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Una sentencia `if` o una construcción `if` ejecuta un bloque de código si una expresión es `true`. Hay algunas variaciones en la construcción `if` básica. Sin embargo, la más simple es la siguiente:

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

donde:

- `boolean_expression` es una combinación de operadores relacionales, operadores condicionales y valores cuyo resultado es un valor `true` o `false`.
- `code_block` representa las líneas de código que se ejecutan si la expresión es `true`.

En primer lugar, se prueba `boolean_expression`. Si la expresión es `true`, se ejecutará el bloque de código. Si `boolean_expression` no es `true`, el programa omitirá la llave que marca el final del bloque de código de la construcción `if`.

Construcción `if`: Ejemplo

```
...
public void goDown() {
    if (currentFloor == MIN_FLOORS) {
        System.out.println("Cannot Go down");
    }
    if (currentFloor > MIN_FLOORS) {
        System.out.println("Going down one floor.");
        currentFloor--;
        System.out.println("Floor: " + currentFloor);
    }
}
```

El ascensor no puede bajar y se muestra un error.

El ascensor puede bajar y se muestra la planta actual más la nueva planta.

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

La clase `ElevatorTest` prueba un objeto `Elevator` mediante la llamada a sus métodos. Uno de los primeros métodos que llama la clase `ElevatorTest` es el método `goDown`. Dos sentencias `if` pueden solucionar este problema. El siguiente método `goDown` contiene dos construcciones `if` que determinan si el ascensor debe bajar o mostrar un error. La clase `ElevatorTest` es la siguiente:

```

public class IfElevator {

    public boolean doorOpen=false; // Default setting
    public int currentFloor = 1; // Default starting point
    public final int TOP_FLOOR = 10;
    public final int MIN_FLOORS = 1;

    public void openDoor() {
        System.out.println("Opening door.");
        doorOpen = true;
        System.out.println("Door is open.");
    }
    public void closeDoor() {
        System.out.println("Closing door.");
        doorOpen = false;
        System.out.println("Door is closed.");
    }
    public void goUp() {
        System.out.println("Going up one floor.");
        currentFloor++;
        System.out.println("Floor: " + currentFloor);
    }
    public void goDown() {

        if (currentFloor == MIN_FLOORS) {
            System.out.println("Cannot Go down");
        }
        if (currentFloor > MIN_FLOORS) {
            System.out.println("Going down one floor.");
            currentFloor--;
            System.out.println("Floor: " + currentFloor);
        }
    }
}

```

Construcción if: Salida

Ejemplo de posible salida:

```
Opening door.  
Door is open.  
Closing door.  
Door is closed.  
Cannot Go down ← La lógica de Elevator evita el problema.  
Going up one floor.  
Floor: 2  
Going up one floor.  
Floor: 3  
...
```

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Sentencias `if` anidadas

```
...
public void goDown() {

    if (currentFloor == MIN_FLOORS) {
        System.out.println("Cannot Go down");
    }

    if (currentFloor > MIN_FLOORS) {

        if (!doorOpen) {
            System.out.println("Going down one floor.");
            currentFloor--;
            System.out.println("Floor: " + currentFloor);
        }
    }
}
}
```

Sentencia
`if` anidada

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

A veces puede que necesite ejecutar una sentencia `if` como parte de otra sentencia `if`. El ejemplo de código ilustra cómo utilizar sentencias `if` anidadas para comprobar los valores de dos variables. Si el valor de la variable `currentFloor` es igual a la constante `MIN_FLOORS`, se muestra un mensaje de error y el ascensor no baja. Si el valor de la variable `currentFloor` es mayor que la constante `MIN_FLOORS` y las puertas se cierran, el ascensor baja. El código de ejemplo de `NestedIfElevator` es el siguiente.

Nota: utilice construcciones `if/else` anidadas con moderación ya que su depuración puede ser confusa.

```

public class NestedIfElevator {

    public boolean doorOpen=false; // Doors are closed by default
    public int currentFloor = 1; // All elevators start on first floor
    public final int TOP_FLOOR = 10;
    public final int MIN_FLOORS = 1;

    public void openDoor() {
        System.out.println("Opening door.");
        doorOpen = true;
        System.out.println("Door is open.");
    }

    public void closeDoor() {
        System.out.println("Closing door.");
        doorOpen = false;
        System.out.println("Door is closed.");
    }

    public void goUp() {
        System.out.println("Going up one floor.");
        currentFloor++;
        System.out.println("Floor: " + currentFloor);
    }

    public void goDown() {
        if (currentFloor == MIN_FLOORS) {
            System.out.println("Cannot Go down");
        }
        if (currentFloor > MIN_FLOORS) {
            if (!doorOpen) {
                System.out.println("Going down one floor.");
                currentFloor--;
                System.out.println("Floor: " + currentFloor);
            }
        }
    }
}

```


Construcción if/else

Sintaxis:

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

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

A menudo, desea que se ejecute un bloque de código si la expresión es true y que se ejecute otro bloque de código si la expresión es false. Puede utilizar una construcción `if` para ejecutar un bloque de código si la expresión es true con una construcción `else` que solo se ejecute si la expresión es false. En el ejemplo de la diapositiva se muestra la sintaxis de una construcción `if/else`, donde:

- `boolean_expression` es una combinación de operadores relacionales, operadores condicionales y valores cuyo resultado es un valor true o false.
- `code_block1` representa las líneas de código que se ejecutan si la expresión es true y `code_block2` representa las líneas de código que se ejecutan si la expresión es false.

Construcción if/else: Ejemplo

```
public void goUp() {
    System.out.println("Going up one floor.");
    currentFloor++;
    System.out.println("Floor: " + currentFloor);
}

public void goDown() {
    if (currentFloor == MIN_FLOORS) {
        System.out.println("Cannot Go down");
    }
    else {
        System.out.println("Going down one floor.");
        currentFloor--;
        System.out.println("Floor: " + currentFloor);
    }
}
}
```

Se ejecuta si la expresión es true.

Se ejecuta si la expresión es false.

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Puede utilizar una sentencia if/else para solucionar el problema del ascensor que va a una planta no válida. El método goDown que aparece en el ejemplo de la diapositiva contiene una construcción if/else que determina si el ascensor debe bajar o mostrar un error. Si el valor de la variable currentFloor es igual a la constante MIN_FLOORS, se muestra un mensaje de error y el ascensor no baja. En caso contrario (else), se supone que el valor de la variable currentFloor es mayor que la constante MIN_FLOORS y el ascensor baja. El ejemplo de código completo es el siguiente:

```

public class IfElseElevator {
    public boolean doorOpen=false; // Default setting
    public int currentFloor = 1; // Default setting
    public final int TOP_FLOOR = 10;
    public final int MIN_FLOORS = 1;

    public void openDoor() {
        System.out.println("Opening door.");
        doorOpen = true;
        System.out.println("Door is open.");
    }
    public void closeDoor() {
        System.out.println("Closing door.");
        doorOpen = false;
        System.out.println("Door is closed.");
    }

    public void goUp() {
        System.out.println("Going up one floor.");
        currentFloor++;
        System.out.println("Floor: " + currentFloor);
    }

    public void goDown() {
        if (currentFloor == MIN_FLOORS) {
            System.out.println("Cannot Go down");
        }
        else {
            System.out.println("Going down one floor.");
            currentFloor--;
            System.out.println("Floor: " + currentFloor);}
        }
    }
}

```

Construcción if/else

Ejemplo de posible salida:

```
Opening door.  
Door is open.  
Closing door.  
Door is closed.  
Cannot Go down ← La lógica de Elevator evita el problema.  
Going up one floor.  
Floor: 2  
Going up one floor.  
Floor: 3  
...
```

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Temas

- Uso de operadores relacionales y condicionales
- Creación de construcciones `if` e `if/else`
- **Encadenamiento de una sentencia `if/else`**
- Uso de una sentencia `switch`

ORACLE®

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Encadenamiento de construcciones `if/else`

Sintaxis:

```
if (boolean_expression) {  
    <code_block1>  
} // end of if construct  
  
else if (boolean_expression){  
    <code_block2>  
} // end of else if construct  
  
else {  
    <code_block3>  
}  
// program continues here
```

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Puede encadenar construcciones `if` y `else` juntas para indicar distintos resultados para varias expresiones diferentes. En el ejemplo de la diapositiva se muestra la sintaxis de una construcción `if/else` encadenada, donde:

- `boolean_expression` es una combinación de operadores relacionales, operadores condicionales y valores cuyo resultado es un valor `true` o `false`.
- `code_block1` representa las líneas de código que se ejecutan si la expresión es `true`.
- `code_block2` representa las líneas de código que se ejecutan si la expresión es `false` y la condición de la segunda sentencia `if` es `true`.
- `code_block3` representa las líneas de código que se ejecutan si la expresión de la segunda sentencia `if` también se evalúa como `false`.

Encadenamiento de construcciones if/else

```

...
public void calculateNumDays() {

    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.");
    }

}
...

```

1 Se ejecuta cuando la sentencia if es true.

2 Se ejecuta cuando la primera sentencia if es false y la sentencia else es true.

3 Se ejecuta cuando la primera sentencia if es false, la primera sentencia else es false y esta sentencia else es true.

4 Se ejecuta cuando todas las sentencias son false.

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

El ejemplo es una clase `IfElseDate` que contiene varias construcciones `if/else` encadenadas que determinan cuántos días hay en un mes. El método `calculateNumDays` encadena tres sentencias `if/else` juntas para determinar el número de días de un mes. Aunque este código es sintácticamente correcto, el encadenamiento de sentencias `if/else` puede dar lugar a un código confuso y se debe evitar.

Temas

- Uso de operadores relacionales y condicionales
- Creación de construcciones `if` e `if/else`
- Encadenamiento de una sentencia `if/else`
- **Uso de una sentencia `switch`**

ORACLE®

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Uso de la construcción switch

Sintaxis:

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

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Otra palabra clave que se utiliza en la toma de decisiones es la palabra clave `switch`. La construcción `switch` ayuda a evitar código confuso porque simplifica la organización de las distintas ramas de código que se pueden ejecutar.


El ejemplo de la clase `IfElseDate` se puede volver a escribir mediante una construcción `switch`. En la diapositiva se muestra la sintaxis de la construcción `switch`, donde:

- La palabra clave `switch` indica una sentencia `switch`.
- `variable` es la variable cuyo valor desea probar. `variable` solo puede ser de tipo `char`, `byte`, `short`, `int` o `String`.
- La palabra clave `case` indica un valor que está probando. Una combinación de la palabra clave `case` y `literal_value` se denomina *etiqueta case*.

- `literal_value` es cualquier valor válido que puede contener una variable. Puede tener una etiqueta `case` para cada valor que desee probar. Los valores literales no pueden ser variables, expresiones, `String` ni llamadas a métodos. Los valores literales pueden ser constantes (variables finales como `MAX_NUMBER` definidas en otra parte), literales (como `'A'` o `10`) o ambos.
- La sentencia `[break;]` es una palabra clave opcional que hace que el flujo de código salga inmediatamente de la sentencia `switch`. Sin una sentencia `break`, se ejecutan todas las sentencias `code_block` que siguen a la sentencia `case` aceptada (hasta que se alcance una sentencia `break` o el final de la construcción `switch`).

Uso de la construcción switch: Ejemplo

```
public class SwitchDate {  
  
    public int month = 10;  
  
    public void calculateNumDays() {  
  
        switch(month) {  
        case 1:  
            case 3:  
            case 5:  
            case 7:  
            case 8:  
            case 10:  
            case 12:  
                System.out.println("There are 31 days in that month.");  
                break;  
        ...  
    }  
}
```

**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

El ejemplo contiene una clase `SwitchDate` que utiliza una construcción `switch` para determinar cuántos días hay en un mes.

El método `calculateNumDays` de la clase `SwitchDate` utiliza una sentencia `switch` para distribuir en el valor de la variable de mes. Si la variable de mes es igual a 1, 3, 5, 7, 8, 10 o 12, el código salta a la etiqueta `case` adecuada y, a continuación, se despliega para ejecutar `System.out.println("There are 31 days in that month.")`.

```
public class SwitchDate {

    public int month = 10;

    public void calculateNumDays() {

        switch(month) {
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:
            System.out.println("There are 31 days in that month.");
            break;
        case 2:
            System.out.println("There are 28 days in that month.");
            break;
        case 4:
        case 6:
        case 9:
        case 11:
            System.out.println("There are 30 days in that month.");
            break;
        default:
            System.out.println("Invalid month.");
            break;
        }
    }
}
```

Cuándo utilizar construcciones `switch`

- Pruebas de igualdad
- Pruebas en un *único* valor, como `customerStatus`
- Pruebas en el valor del tipo `int`, `short`, `byte`, `o char` y `String`
- Pruebas en un valor corregido conocido en el momento de la compilación

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Prueba

¿Cuál es el objetivo del bloque `else` en una sentencia `if/else`?

- a. Incluir el resto del código de un método.
- b. Incluir código que se ejecuta cuando la expresión de una sentencia `if` es `false`.
- c. Probar si una expresión es `false`.

ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Respuesta: b

Prueba

¿Cuál de las siguientes afirmaciones es adecuada para probar un valor de una construcción `switch`?

- a. La construcción `switch` prueba si los valores son mayores o menores que un único valor.
- b. La construcción `switch` se prueba en una única variable.
- c. La construcción `switch` prueba el valor de un tipo de dato `float`, `double` o `boolean` y `String`.



Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Respuesta: b

- La respuesta a es incorrecta porque se debe utilizar un operador relacional para probar si los valores son mayores o menores que un único valor.
- La respuesta b es correcta.
- La respuesta c es incorrecta. La construcción `switch` prueba el valor de los tipos `char`, `byte`, `short`, `int` o `String`.

Resumen

En esta lección, debe haber aprendido lo siguiente:

- Utilizar un operador relacional
- Probar la igualdad entre cadenas
- Utilizar un operador condicional
- Crear construcciones `if` e `if/else`
- Anidar una sentencia `if`
- Encadenar una sentencia `if/else`
- Utilizar una sentencia `switch`



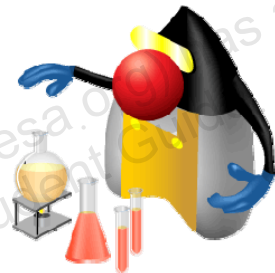
ORACLE

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Visión general de la práctica 7-1: Escritura de una clase que utiliza la sentencia `if/else`

En esta práctica, creará clases que utilicen construcciones `if` e `if/else`. Existen dos secciones en esta práctica:

- En la primera sección, creará la clase `DateTwo` que utiliza sentencias `if/else` para mostrar el día de la semana según el valor de la variable.
- En la segunda sección, creará la clase `Clock` que utiliza sentencias `if/else` para mostrar la parte del día, según la hora del día.

**ORACLE**

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Visión general de la práctica 7-2: Escritura de una clase que utiliza la sentencia `switch`

En esta práctica, creará una clase llamada `Month` que utilice sentencias `switch` para mostrar el nombre del mes según el valor numérico de un campo.

ORACLE®

Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.