

Conceptos fundamentales de Java SE 7

Volumen II - Guía del Alumno

D67234CS20

Edición 2.0

Noviembre de 2011

D81767

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 <code>getter</code> y <code>setter</code>	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.

8

Creación y uso de matrices

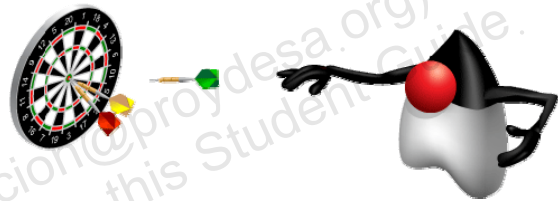
ORACLE®

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

Objetivos

Al finalizar esta lección, debería estar capacitado para:

- Declarar, instanciar e inicializar una matriz unidimensional
- Declarar, instanciar e inicializar una matriz bidimensional
- Acceder a un valor de una matriz
- Describir cómo se almacenan las matrices en memoria
- Declarar e inicializar una ArrayList
- Usar una matriz `args`



ORACLE

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

Temas

- **Visión general de las matrices**
- Declaración, instanciación e inicialización de matrices
- Acceso a los argumentos de la línea de comandos
- Trabajar con matrices bidimensionales
- Trabajar con una ArrayList

ORACLE

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

Introducción a las matrices

- Una matriz es un objeto contenedor que incluye un grupo de valores de un único tipo.
- Un valor de la matriz puede ser un tipo primitivo o un tipo de objeto.
- La longitud de una matriz se establece al crearla.
- Tras la creación, la longitud de una matriz no se puede cambiar.
- Cada elemento de una matriz se denomina *elemento*.
- A cada elemento se accede mediante un índice numérico.
- El índice del primer elemento es 0 (cero).

ORACLE

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

Matrices unidimensionales

Ejemplo:

```
int ageOne = 27;  
int ageTwo = 12;  
int ageThree = 82;  
int ageFour = 70;  
int ageFive = 54;  
int ageSix = 6;  
int ageSeven = 1;  
int ageEight = 30;  
int ageNine = 34;  
int ageTen = 42;
```

ORACLE

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

Imagine un programa en el que almacena las edades de 10 personas. Puede crear variables individuales para incluir cada uno de los 10 valores. Para ello, puede utilizar el código mostrado en la diapositiva, pero hay problemas con este enfoque. ¿Qué sucedería si tuviera que almacenar 1.000 o 10.000 edades? A medida que el número de valores aumenta, el programa es cada vez más difícil de gestionar. O bien, ¿qué sucedería si tuviera que buscar la edad media u ordenar las edades en orden ascendente? Tendría que hacer referencia a cada variable de forma individual en el código.

Como verá, las matrices en Java (y las construcciones relacionadas como listas) ofrecen una forma mucho más cómoda de trabajar con juegos de datos. En esta lección, aprenderá acerca de las matrices. En la lección titulada “Uso de construcciones de bucle”, aprenderá a utilizar bucles para trabajar mediante programación en todos los valores de una matriz.

Creación de matrices unidimensionales

Matriz de tipos `int`

27	12	82	70	54	1	30	34
----	----	----	----	----	---	----	----

Matriz de tipos `Shirt`



Matriz de tipos `String`

Hugh Mongus
Aaron Datires
Stan Ding
Albert Kerkie
Carrie DeKeys
Walter Mellon
Hugh Morris
Moe DeLawn

ORACLE

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

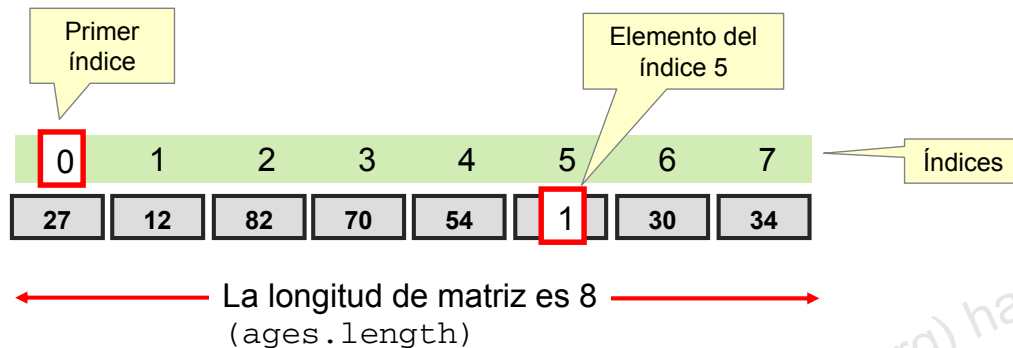
El lenguaje de programación Java permite agrupar varios valores del mismo tipo (listas) mediante matrices. Las matrices resultan útiles cuando tiene porciones de datos relacionadas (como las edades de varias personas), pero no desea crear variables independientes para incluir cada una de estas porciones.

Puede crear una matriz de tipos primitivos, como `int` o una matriz de referencias a tipos de objetos, como `Shirt` o `String`. Cada parte de la matriz es un elemento. Si declara una matriz de 100 tipos `int`, hay 100 elementos. Puede acceder a cada elemento concreto de la matriz mediante su ubicación o índice en la matriz.

El diagrama de la diapositiva muestra ejemplos de matrices para tipos `int`, tipos `Shirt` y tipos `String`.

Índices y longitud de matriz

Matriz ages de ocho elementos



ORACLE

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

Una matriz es un objeto contenedor que incluye un número fijo de valores de un único tipo. La longitud de una matriz se establece al crearla. Tras la creación, la longitud de una matriz no se puede cambiar.

Cada elemento de una matriz se denomina *elemento* y a cada elemento se accede mediante su índice numérico. Como se muestra en el diagrama de la diapositiva, la numeración empieza por 0. Por ejemplo, al elemento ocho se accede en el índice 7.

Se puede acceder a la longitud de una matriz mediante una notación de puntos para acceder al campo `length`. Suponiendo que la matriz del diagrama se denomina `ages`, puede utilizar: `int agesLength = ages.length;`

De esta forma, se asigna un valor 8 a `int agesLength`.

Temas

- Visión general de las matrices
- **Declaración, instanciación e inicialización de matrices**
- Acceso a los argumentos de la línea de comandos
- Trabajar con matrices bidimensionales
- Trabajar con una ArrayList

ORACLE

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

Declaración de una matriz unidimensional

- Sintaxis:

```
type [] array_identifier;
```

- Declarar matrices de tipos `char` e `int`:

```
char [] status;
int [] ages;
```

- Declarar matrices de referencias de objetos de tipo `Shirt` y `String`:

```
Shirt [] shirts;
String [] names;
```



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

Las matrices se manejan mediante un objeto `Array` implícito (que no está disponible en la API de Java). Al igual que con cualquier objeto, debe declarar una referencia de objeto a la matriz, instanciar un objeto `Array` y, a continuación, inicializar el objeto `Array` para poder utilizarlo.

La sintaxis utilizada para declarar una matriz unidimensional es la siguiente:

```
type [] array_identifier;
```

donde:

- `type` representa el tipo de dato primitivo o el tipo de objeto para los valores almacenados en la matriz.
- `[]` informa al compilador de que está declarando una matriz.
- `array_identifier` es el nombre asignado para hacer referencia a la matriz.

Al declarar una matriz, el compilador y Java Virtual Machine (JVM) no conocen el tamaño que tendrán las matrices ya que ha declarado una variable de referencia que actualmente no apunta a ningún objeto.

Instanciación de una matriz unidimensional

- Sintaxis:

```
array_identifier = new type [length];
```

- Ejemplos:

```
status = new char [20];
ages = new int [5];

names = new String [7];
shirts = new Shirt [3];
```



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

Para poder inicializar una matriz, debe instanciar un objeto Array lo suficientemente grande para incluir todos los valores de la matriz. Instancie una matriz mediante la definición del número de elementos de la matriz.

La sintaxis utilizada para instanciar un objeto Array es la siguiente:

```
array_identifier = new type [length];
```

donde:

- `array_identifier` es el nombre asignado para hacer referencia a la matriz.
- `type` representa el tipo de dato primitivo o el tipo de objeto para los valores almacenados en la matriz.
- `length` representa el tamaño (en número de elementos) de la matriz.

Al instanciar un objeto Array, cada elemento primitivo se inicializa con el valor cero para el tipo especificado. En el caso de la matriz `char` denominada `status`, cada valor se inicializa con `\u0000` (el carácter nulo del juego de caracteres Unicode). Para la matriz `int` denominada `ages`, el valor inicial es el valor entero 0. Para las matrices `names` y `shirt`, las referencias de objetos se inicializan con el valor nulo.

Inicialización de una matriz unidimensional

- **Sintaxis:**

```
array_identifier[index] = value;
```

- **Definir valores en la matriz ages:**

```
ages[0] = 19;
ages[1] = 42;
ages[2] = 92;
ages[3] = 33;
```

- **Definir referencias en los objetos Shirt de la matriz shirts:**

```
shirts[0] = new Shirt();
shirts[1] = new Shirt();
shirts[2] = new Shirt();
```



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

Puede rellenar el contenido de una matriz después de haberla creado. La sintaxis para definir los valores de una matriz es la siguiente:

```
array_identifier[index] = value;
```

donde:

- `array_identifier` es el nombre asignado a la matriz.
- `index` representa la ubicación de la matriz en la que se colocará el valor.

Utilice la palabra clave `new` para crear los objetos `Shirt` y colocar las referencias a los objetos `Shirt` en cada una de las posiciones de la matriz.

Nota: el índice del primer elemento de una matriz es 0 y el del último es la longitud de la matriz menos 1. Por ejemplo, el último elemento de una matriz de seis elementos es el índice 5.

Declaración, instanciación e inicialización de matrices unidimensionales

- Sintaxis:

```
type [] array_identifier = {comma-separated list of values or expressions};
```

- Ejemplos:

```
int [] ages = {19, 42, 92, 33, 46};
Shirt [] shirts = {new Shirt(), new Shirt(), new Shirt()};
```

- No permitida (NetBeans mostrará un error):

```
int [] ages;
ages = {19, 42, 92, 33, 46};
```

ORACLE

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

Si sabe los valores que desea que estén en la matriz en el momento de la declaración, puede declarar, instanciar y definir los valores para un objeto Array en la misma línea de código. La sintaxis para ello es la siguiente:

```
type [] array_identifier =
    {comma-separated_list_of_values_or_expressions};
```

donde:

- `type` representa el tipo de dato primitivo o el tipo de objeto para los valores que se van a almacenar.
- `[]` informa al compilador de que está declarando una matriz.
- `array_identifier` es el nombre asignado a la matriz.
- `{comma-separated_list_of_values_or_expressions}` representa una lista de valores que desea almacenar en la matriz.

En los ejemplos de la diapositiva se muestran sentencias que combinan la declaración, instanciación e inicialización. Observe cómo se utiliza la palabra clave `new` para instanciar el objeto `Shirt` de forma que se pueda colocar una referencia a dicho objeto en la matriz.

El ejemplo final de la diapositiva devuelve un error. No puede declarar e inicializar una matriz en líneas separadas mediante la técnica de lista separada por comas.

Acceso a un valor de una matriz

- Definición de un valor:

```
status[0] = '3';  
names[1] = "Fred Smith";  
ages[1] = 19;  
prices[2] = 9.99F;
```

- Obtención de un valor:

```
char s = status[0];  
String name = names [1];  
int age = ages[1];  
double price = prices[2];
```

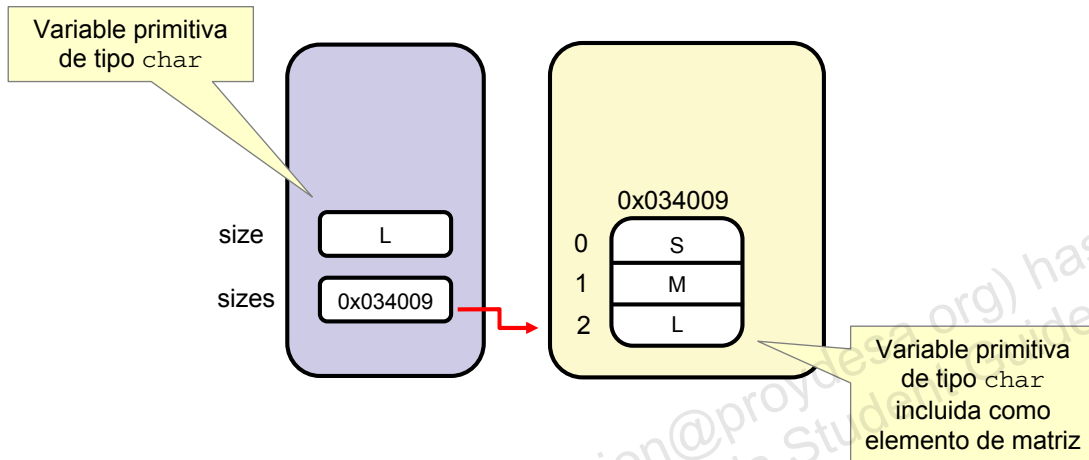
ORACLE

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

A cada elemento de una matriz se accede mediante su índice. Para acceder a un valor de la matriz, indique el nombre de la matriz y el número de índice del elemento (entre corchetes []) a la derecha de un operador de asignación.

Almacenamiento de matrices en memoria

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



ORACLE

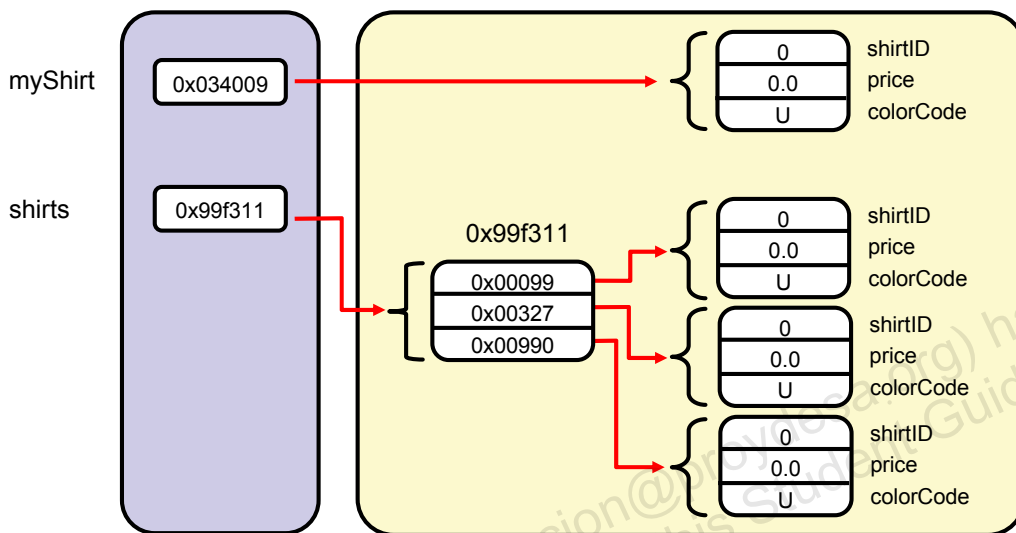
Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Las matrices son objetos a los que hace referencia una variable de referencia de objeto. En el diagrama de la diapositiva se ilustra el almacenamiento en memoria de una matriz primitiva en comparación con el almacenamiento en memoria de un tipo de dato primitivo.

El valor de la variable `size` (un primitivo `char`) es `L`. El valor de `sizes[]` es `0x334009` y apunta a un objeto de tipo `array` (de tipos `char`) con tres valores. El valor de `sizes[0]` es `char S`, el valor de `sizes[1]` es `char M` y el valor de `sizes[2]` es `char L`.

Almacenamiento de matrices de referencias en memoria

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



ORACLE®

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

En el diagrama de la diapositiva se ilustra el almacenamiento en memoria de una matriz de referencias de objetos. El valor de la referencia de objeto `myShirt` es `x034009`, que es una dirección a un objeto de tipo `Shirt` con los valores 0, 0.0 y U. El valor de la referencia de objeto `shirts[]` es `x99f311`, que es una dirección a un objeto de tipo `array` (de referencias de objetos `Shirt`) que contiene tres referencias de objetos:

- El valor del índice `shirts[0]` es `0x00099`, que es una referencia de objeto que apunta a un objeto de tipo `Shirt`.
- El valor del índice `shirts[1]` es `0x00327`, que es una referencia de objeto que apunta a otro objeto de tipo `Shirt`.
- El valor del índice `shirts[2]` es `0x00990`, que es una referencia de objeto que apunta a otro objeto de tipo `Shirt`.

Prueba

El siguiente código es la sintaxis correcta para _____ una matriz: `array_identifier = new type [length];`

- a. Declarar
- b. Definir valores de matriz para
- c. Instanciar
- d. Declarar, instanciar y definir valores de matriz para

ORACLE

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

Respuesta: c

Prueba

Dada la siguiente declaración de matriz, determine cuáles de las tres afirmaciones que aparecen a continuación son verdaderas.

```
int [ ] autoMobile = new int [13];
```

- a. `autoMobile[0]` es la referencia al primer elemento de la matriz.
- b. `autoMobile[13]` es la referencia al último elemento de la matriz.
- c. Hay 13 enteros en la matriz `autoMobile`.

ORACLE

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

Respuesta: a, c

Temas

- Visión general de las matrices
- Declaración, instanciación e inicialización de matrices
- **Acceso a los argumentos de la línea de comandos**
- Trabajar con matrices bidimensionales
- Trabajar con una ArrayList

ORACLE

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

Uso de la matriz `args` en el método `main`

- Los parámetros se pueden escribir en la línea de

```
> java ArgsTest Hello World!
args[0] is Hello
args[1] is World!
```

- Código para recuperar los parámetros:

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

ORACLE

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

Al transferir cadenas al programa en la línea de comandos, las cadenas se colocan en la matriz `args`. Para utilizar estas cadenas, debe extraerlas de la matriz `args` y, opcionalmente, convertirlas en su tipo correcto (porque la matriz `args` es de tipo `String`).

La clase `ArgsTest` mostrada en la diapositiva extrae dos argumentos `String` transferidos en la línea de comandos y los muestra.

Para agregar parámetros en la línea de comandos, debe dejar uno o más espacios después del nombre de la clase (en este caso, `ArgsTest`) y uno o más espacios entre cada parámetro agregado.

NetBeans no permite ejecutar una clase Java desde la línea de comandos, pero puede definir argumentos de la línea de comandos como una propiedad del proyecto en el que está el código. Utilizará esta técnica en la práctica de esta lección.

Conversión de argumentos String en otros tipos

- Los números se pueden introducir como parámetros:

```
> java ArgsTest 2 3
Total is: 23
Total is: 5
```

Concatenación, no adición.

- Conversión de String en int:

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

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

Son argumentos String.

Integer.parseInt
Int() se
convierte en int.

Observe los paréntesis.

ORACLE

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

El método `main` trata todo lo que escriba como una cadena de literales. Si desea utilizar la representación de cadena de un número en una expresión, debe convertir la cadena en su equivalente numérico. Cada tipo de dato tiene una clase asociada que contiene métodos de utilidad estáticos para convertir cadenas en ese tipo de dato (clase `Integer` para `int`, clase `Byte` para `byte`, clase `Long` para `long`, etc.). Por ejemplo, para convertir el primer argumento transferido al método `main` en un tipo `int`, utilice `Integer.parseInt(args[0])`.

Tenga en cuenta que son necesarios los paréntesis alrededor de `arg1 + arg2` para que el signo `+` indique adición en lugar de concatenación.

Temas

- Visión general de las matrices
- Declaración, instanciación e inicialización de matrices
- Acceso a los argumentos de la línea de comandos
- **Trabajar con matrices bidimensionales**
- Trabajar con una ArrayList

ORACLE

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

Descripción de matrices bidimensionales

	Domingo	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado
Semana 1							
Semana 2							
Semana 3							
Semana 4							

ORACLE

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

También puede almacenar matrices de datos mediante matrices multidimensionales. Las matrices multidimensionales tienen dos o más dimensiones. Una matriz bidimensional (2D) es una matriz de matrices, una matriz 3D es una matriz de matrices 2D, una matriz 4D es una matriz de matrices 3D, etc. Una matriz bidimensional es similar a una hoja de cálculo con varias columnas (cada columna representa una matriz o lista de elementos) y varias filas.

El diagrama de la diapositiva muestra una matriz bidimensional. Tenga en cuenta que los nombres descriptivos Semana 1, Semana 2, Lunes, Martes, etc. no se utilizarán para acceder a los elementos de la matriz. En su lugar, Semana 1 será el índice 0 y Semana 4 será el índice 3 en esa dimensión, mientras que Domingo será el índice 0 y Sábado el índice 6 en la otra dimensión.

Declaración de una matriz bidimensional

- Sintaxis:

```
type [][] array_identifier;
```

- Ejemplo:

```
int [][] yearlySales;
```

ORACLE

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

Las matrices bidimensionales necesitan un juego adicional de corchetes. El proceso de creación y uso de matrices bidimensionales es por lo demás igual que el de las matrices unidimensionales. La sintaxis para declarar una matriz bidimensional es la siguiente:

```
type [][] array_identifier;
```

donde:

- `type` representa el tipo de dato primitivo o el tipo de objeto para los valores almacenados en la matriz.
- `[][]` informa al compilador de que está declarando una matriz bidimensional.
- `array_identifier` es el nombre asignado a la matriz durante la declaración.

En el ejemplo mostrado, se declara una matriz bidimensional (una matriz de matrices) denominada `yearlySales`.

Instanciación de una matriz bidimensional

- Sintaxis:

```
array_identifier = new type [number_of_arrays] [length];
```

- Ejemplo:

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

	Trimestre 1	Trimestre 2	Trimestre 3	Trimestre 4
Año 1				
Año 2				
Año 3				
Año 4				
Año 5				

ORACLE

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

La sintaxis para instanciar una matriz bidimensional es la siguiente:

```
array_identifier = new type [number_of_arrays] [length];
```

donde:

- `array_identifier` es el nombre asignado a la matriz durante la declaración.
- `number_of_arrays` es el número de matrices de la matriz.
- `length` es la longitud de cada matriz de la matriz.

En el ejemplo mostrado en la diapositiva, se instancia una matriz de matrices para cantidades de ventas trimestrales a lo largo de cinco años. La matriz `yearlySales` contiene cinco elementos de la matriz de tipo `int` (cinco submatrices). Cada submatriz tiene un tamaño de cuatro elementos y realiza un seguimiento de las ventas de un año en cuatro trimestres.

Inicialización de una matriz bidimensional

Ejemplo:

```
yearlySales[0][0] = 1000;
yearlySales[0][1] = 1500;
yearlySales[0][2] = 1800;
yearlySales[1][0] = 1000;
yearlySales[3][3] = 2000;
```

	Trimestre 1	Trimestre 2	Trimestre 3	Trimestre 4
Año 1	1000	1500	1800	
Año 2	1000			
Año 3				
Año 4				2000
Año 5				

ORACLE

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

Al definir (u obtener) valores de una matriz bidimensional, indique el número de índice en la matriz mediante un número que represente la fila, seguido de un número que represente la columna. En el ejemplo de la diapositiva, se muestran cinco asignaciones de valores a elementos de la matriz `yearlySales`.

Temas

- Visión general de las matrices
- Declaración, instanciación e inicialización de matrices
- Acceso a los argumentos de la línea de comandos
- Trabajar con matrices bidimensionales
- Trabajar con una ArrayList

ORACLE

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

Clase ArrayList

Las matrices no son la única forma de almacenar listas de datos relacionados:

- ArrayList es una de una serie de clases de lista.
- Tiene un juego de métodos útiles para gestionar sus elementos:
 - `add()`, `get()`, `remove()`, `indexOf()` y muchos otros
- No necesita especificar el tamaño al instanciar una ArrayList:
 - A medida que agregue más elementos, la ArrayList aumentará según sea necesario.
 - Puede especificar una capacidad inicial, pero no es obligatorio hacerlo.
- Una ArrayList solo puede almacenar objetos, no primitivos.

ORACLE

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

Para las listas que son muy dinámicas, puede resultar más sencillo trabajar con un objeto de tipo List especializado. Éste puede evitar que tenga que escribir código para:

- Realizar un seguimiento del índice de la última porción de datos agregada.
- Realizar un seguimiento de hasta dónde está llena la matriz y determinar si es necesario cambiar el tamaño.
- Aumentar el tamaño de la matriz mediante la creación de una nueva y la copia de los elementos de la actual en otra.

Nombres de clases y sentencia de importación

- ArrayList está en el paquete `java.util`.
- Para hacer referencia a la ArrayList de su código, puede cualificar totalmente

```
java.util.ArrayList myList;
```

o bien puede agregar la sentencia de importación en la parte superior de la clase.

```
import java.util.ArrayList;
public class ArrayListExample {
    public static void main (String[] args) {
        ArrayList myList;
    }
}
```



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

Las clases del lenguaje de programación Java se agrupan en paquetes según su funcionalidad. Por ejemplo, todas las clases relacionadas con el lenguaje de programación Java principal están en el paquete `java.lang`, que contiene clases que son fundamentales para el lenguaje de programación Java, como `String`, `Math` e `Integer`. Se puede hacer referencia a las clases del paquete `java.lang` en el código simplemente mediante sus nombres de clases. No necesitan una cualificación completa ni el uso de una sentencia de importación.

Todas las clases de otros paquetes (por ejemplo, `ArrayList`) necesitan que las cualifique totalmente en el código, o bien que utilice una sentencia de importación para que se pueda hacer referencia a ellas directamente en el código.

La sentencia de importación puede ser la siguiente:

- Solo para la clase en cuestión
`java.util.ArrayList;`
- Para todas las clases del paquete
`java.util.*;`

Trabajar con una ArrayList

```

ArrayList myList;

myList = new ArrayList();

myList.add("John");
myList.add("Ming");
myList.add("Mary");
myList.add("Prashant");
myList.add("Desmond");

myList.remove(0);
myList.remove(myList.size()-1);
myList.remove("Mary");

System.out.println(myList);

```

Declarar una referencia.

Instanciar la ArrayList.

Inicializar la ArrayList.

Modificar la ArrayList.

ORACLE

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

La declaración de una ArrayList es exactamente igual que la declaración de cualquier otro tipo de referencia. Asimismo, la instanciación de una ArrayList es igual que la instanciación de cualquier otro objeto. (Puede consultar la documentación para ver otras posibilidades de instanciación).

Existe una serie de métodos para agregar datos a la ArrayList. En el ejemplo de la diapositiva se utiliza el más simple, `add()`, para agregar una cadena. Cada llamada a `add` agrega un nuevo elemento al final de la ArrayList.

Finalmente, una gran ventaja de ArrayList con respecto a una matriz es que hay muchos métodos disponibles para manipular los datos. En este ejemplo se muestra solo un método, pero es muy potente.

- `remove(0)`: elimina el primer elemento (en este caso, "John").
- `remove(myList.size() - 1)`: elimina el último elemento debido a que `myList.size()` proporciona el número de elementos de la matriz, por lo que el último es el tamaño menos 1 (elimina "Desmond").
- `remove("Mary")`: elimina un elemento concreto. En este caso, tiene la comodidad de hacer referencia no a la ubicación del elemento en la ArrayList, sino al elemento en sí.

Puede transferir una ArrayList a `System.out.println()` y la salida resultante será la siguiente:

```
[Ming, Prashant]
```

Prueba

Una matriz bidimensional es similar a _____.

- a. Una lista de la compra
- b. Una lista de tareas
- c. Una matriz
- d. Un gráfico de barras que contiene las dimensiones de varios cuadros

ORACLE

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

Respuesta: c

Resumen

En esta lección debe haber aprendido lo siguiente:

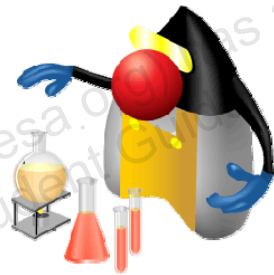
- Una matriz en Java es un tipo de dato que consta de un juego de otros tipos de dato:
 - Los tipos de dato pueden ser objetos o primitivos.
 - Cada valor de dato es un elemento de la matriz.
- Las matrices se crean con un tamaño concreto (número de elementos).
- A cada elemento de una matriz se accede mediante su índice:
 - El primer índice es 0 (cero).
- El tipo de dato de una matriz puede ser otra matriz:
 - Esto crea una matriz bidimensional.
- Otra opción es utilizar una clase List especializada, como ArrayList.



ORACLE

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

En esta práctica, creará una matriz que contenga el número de días de vacaciones que recibe un empleado de la compañía Duke's Choice.



ORACLE®

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

Visión general de la práctica 8-2: Creación y trabajo con una ArrayList

En esta práctica, experimentará con el relleno y la manipulación de ArrayLists. En esta práctica, podrá:

- Crear dos clases, `NamesList` y `NamesListTest`
- Agregar un método a la clase `NamesList` para rellenar la lista y mostrar su contenido
- Agregar un método para manipular los valores de la lista

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.

Visión general de la práctica 8-3: Uso de argumentos de tiempo de ejecución y análisis de la matriz args

En esta práctica, escribirá un juego de adivinanzas que acepte un argumento y muestre un mensaje asociado. En esta práctica, podrá:

- Crear una clase que acepte un argumento de tiempo de ejecución
- Generar un número aleatorio
- Comparar el número aleatorio con un valor de argumento

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.

9 Uso de construcciones de bucle

ORACLE®

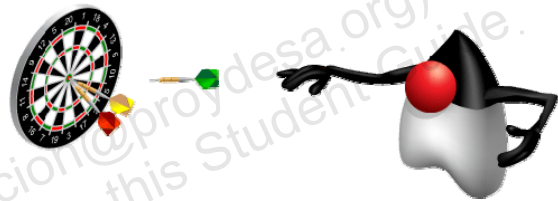
Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

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

Objetivos

Al finalizar esta lección, debería estar capacitado para:

- Crear un bucle `while`
- Anidar un bucle `while`
- Desarrollar y anidar un bucle `for`
- Codificar y anidar un bucle `do/while`
- Utilizar una `ArrayList` en un bucle `for`
- Comparar construcciones de bucle



ORACLE

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

Temas

- Creación de un bucle `while`
- Desarrollo de un bucle `for`
- Anidamiento de un bucle `for` y un bucle `while`
- Uso de una matriz en un bucle `for`
- Codificación y anidamiento de un bucle `do/while`
- Comparación de construcciones de bucle

ORACLE

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

Bucles

Los bucles se utilizan frecuentemente en programas para repetir bloques de sentencias hasta que una expresión es false.

Hay tres tipos principales de bucles:

- Bucle `while`: se repite hasta que una expresión es true.
- Bucle `do/while`: se ejecuta una vez y, a continuación, se sigue repitiendo mientras es true.
- Bucle `for`: se repite un número definido de veces.

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.

Comportamiento de repetición



```
while (!areWeThereYet) {  
  
    read book;  
    argue with sibling;  
    ask, "Are we there yet?";  
  
}  
  
Woohoo!;  
Get out of car;
```

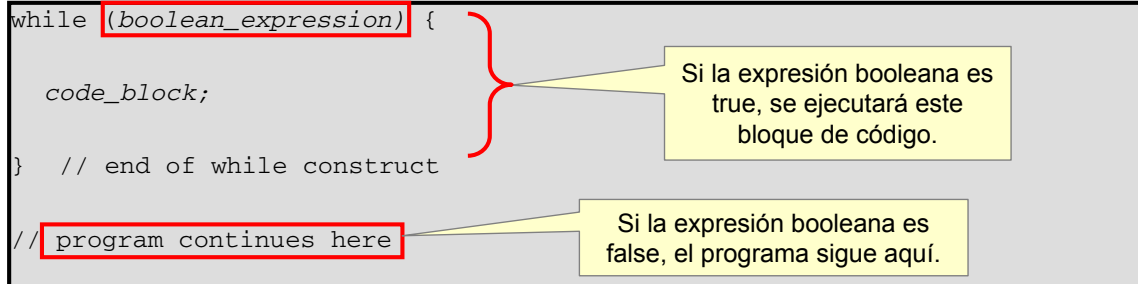
ORACLE

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

En la programación informática, es normal necesitar repetir un número de sentencias. Normalmente, el código sigue repitiendo las sentencias hasta que algo cambia. A continuación, el código salta fuera del bucle y continúa con la siguiente sentencia.

Creación de bucles while

Sintaxis:

**ORACLE**

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

Bucle while en Elevator

```
public void setFloor() {  
    // Normally you would pass the desiredFloor as an argument to the  
    // setFloor method. However, because you have not learned how to  
    // do this yet, desiredFloor is set to a specific number (5)  
    // below.
```

```
    int desiredFloor = 5;  
    while ( currentFloor != desiredFloor ){  
        if (currentFloor < desiredFloor) {  
            goUp();  
        }  
        else {  
            goDown();  
        }  
    }  
}
```

Si la expresión
booleana devuelve
true, se ejecutará el
bucle while.

ORACLE

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

El código de la diapositiva muestra un bucle `while` muy simple en la clase `Elevator`. Recuerde que este ascensor concreto acepta comandos para subir o bajar solo una planta cada vez. Por lo tanto, para moverse un número de plantas, es necesario llamar al método `goUp()` o `goDown()` un número de veces.

Observe cómo se escribe la expresión booleana. La expresión devuelve `true` si `currentFloor` no es igual a `desiredFloor`. Por lo tanto, cuando estas dos variables son iguales, esta expresión devuelve `false` (porque el ascensor ahora está en la planta deseada) y no se ejecuta el bucle `while`.

Tipos de variables

```

public class Elevator {
    public boolean doorOpen=false;
    public int currentFloor = 1;
    public final int TOP_FLOOR = 10;
    public final int BOTTOM_FLOOR = 1;

    ... < lines of code omitted > ...

    public void setFloor() {
        int desiredFloor = 5;
        while ( currentFloor != desiredFloor ){
            if (currentFloor < desiredFloor) {
                goUp();
            } else {
                goDown();
            }
        } // end of while loop
    } // end of method
} // end of class

```

Variables de instancia (campos)

Variable local

Ámbito de desiredFloor

ORACLE

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

Este método `setFloor` utiliza dos tipos distintos de variables. La variable `currentFloor` es una variable de instancia, normalmente denominada *campo*. Se trata de un miembro de la clase `Elevator`. En una lección anterior, hemos visto cómo se puede acceder a los campos de un objeto mediante la notación de puntos. Los campos se declaran fuera del código del método, normalmente justo después de la declaración de la clase.

La variable `desiredFloor` es una variable local, declarada en el método `setFloor` y accesible solo en dicho método. Otra forma de decir esto es que su ámbito es el método `setFloor`. Como verá más adelante, las variables locales también se pueden declarar en bucles o sentencias `if`. Independientemente de si se declara una variable local en un método, un bucle o una sentencia `if`, su ámbito siempre es el bloque en el que se declara.

Bucle `while`: Ejemplo 1

Ejemplo:

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

Resultado:

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



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

En el ejemplo se muestra un código para generar la raíz cuadrada de un número. La expresión booleana eleva al cuadrado el valor actual de la raíz cuadrada y comprueba si se aproxima al número cuya raíz cuadrada está intentando calcular. Si se aproxima lo suficiente (la expresión devuelve `true`), la ejecución del programa omite las sentencias del bloque `while` y continúa con la sentencia `System.out.println()` que muestra la raíz cuadrada. Si el valor aún no se aproxima lo suficiente, el código del bloque se ejecuta y hace dos cosas:

- Ajusta el valor de `squareRoot` para que se aproxime la siguiente vez que se compruebe
- Muestra el valor “estimado” actual de `squareRoot`

Bucle `while`: Ejemplo 2

Ejemplo:

```
int initialSum = 500;
int interest = 7;           // per cent
int years = 0;
int currentSum = initialSum * 100; // Convert to pennies
while ( currentSum <= 100000 ) {
    currentSum += currentSum * interest/100;
    years++;
    System.out.println("Year " + years + ": " + currentSum/100);
}
```

Comprueba si el dinero se ha duplicado ya.

Si no se ha duplicado, agrega el interés de otro año.

Resultado:

```
... < some results not shown > ...
Year 9: 919
Year 10: 983
Year 11: 1052
```

El bucle `while` se itera 11 veces antes de que la prueba booleana se evalúe en true.

ORACLE

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

En el ejemplo de la diapositiva se muestra cuánto se tardará en duplicar el dinero con un tipo de interés concreto. La expresión booleana del bucle `while` comprueba si el dinero (convertido a centavos) se ha duplicado. Si no lo ha hecho, el bloque del bucle agrega el interés de otro año al total actual y el bucle repite la comprobación de la expresión booleana.

Nota: la conversión a centavos se hace para simplificar el ejemplo de forma que se pueda utilizar el tipo `int`.

Bucle while con contador

Ejemplo:

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

Declarar e inicializar una variable de contador.

Comprobar para ver si el contador ha excedido el valor 4.

Imprimir un asterisco e incrementar el contador.

Salida:

```
/*
 *
 *
 *
 *
 */
```

ORACLE

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

Los bucles se utilizan a menudo para repetir un juego de comandos un número concreto de veces. Puede hacer esto fácilmente mediante la declaración e inicialización de un contador (normalmente de tipo `int`). Para ello, incremente la variable en el bucle y compruebe si el contador ha alcanzado un valor concreto en la expresión booleana `while`.

Aunque esto funciona, Java tiene un bucle de contador especial (un bucle `for`), que se trata en las siguientes diapositivas.

Temas

- Creación de un bucle `while`
- **Desarrollo de un bucle `for`**
- Anidamiento de un bucle `for` y un bucle `while`
- Uso de una matriz en un bucle `for`
- Codificación y anidamiento de un bucle `do/while`
- Comparación de construcciones de bucle

ORACLE

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

Bucle for

Bucle while:

La inicialización de la variable de contador se mueve aquí.

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

El incremento del contador va aquí.

Bucle for:

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

    System.out.println("    *");

}
```

Las expresiones booleanas permanecen aquí.

ORACLE

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

En el bucle `for`, las tres expresiones necesarias para un bucle que se ejecuta un número de veces definido se mueven dentro de los paréntesis después de la palabra clave `for`. Esto permite que el bucle `for` sea más compacto y legible.

Desarrollo de un bucle for

Sintaxis:

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

    code_block;

}
```

Ejemplo:

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

    System.out.println(i + t);

}
```

Las tres
partes
del
bucle for

ORACLE

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

Tenga en cuenta que los bucles `for` son muy versátiles; puede inicializar más de una variable en la primera parte y modificar más de una variable en la tercera parte de la sentencia `for`. Además, el tipo no tiene que ser `int`.

El código de la diapositiva declara dos Strings y, a medida que realiza el bucle, agrega a una String mientras elimina de la otra. Estos cambios están en la tercera parte de la sentencia `for`. Esta parte es para actualizaciones y, aunque a menudo se utiliza para incrementar String, se puede utilizar para cualquier tipo de actualización (como se muestra aquí).

La salida del bucle es la siguiente:

```
|-----
||-----
|||-----
||||-----
|||||-----
||||||-----
```


Temas

- Creación de un bucle `while`
- Desarrollo de un bucle `for`
- **Anidamiento de un bucle `for` y un bucle `while`**
- Uso de una matriz en un bucle `for`
- Codificación y anidamiento de un bucle `do/while`
- Comparación de construcciones de bucle

ORACLE

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

Bucle for anidado

Código:

```
int height = 4;
int width = 10;

for (int rowCount = 0; rowCount < height; rowCount++ ) {

    for (int colCount = 0; colCount < width; colCount++ ) {
        System.out.print("@");
    }
    System.out.println();
}
```

ORACLE

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

El código de la diapositiva muestra un bucle anidado simple para generar la salida de un bloque de símbolos @ con la altura y el ancho indicados en las variables locales iniciales. Observe cómo el código externo imprime una nueva línea para empezar una nueva fila, mientras que el bucle interno utiliza el método `print()` de `System.out` para imprimir un símbolo @ para cada columna.

Bucle `while` anidado

Código:

```
String name = "Lenny";
String guess = "";
int numTries = 0;

while (!guess.equals(name.toLowerCase())) {
    guess = "";
    while (guess.length() < name.length()) {
        char asciiChar = (char)(Math.random() * 26 + 97);
        guess = guess + asciiChar;
    }
    numTries++;
}
System.out.println(name + " found after " + numTries + " tries!");
```



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

Éste es un bucle `while` anidado que es un poco más complejo que el ejemplo de `for` anterior. El bucle anidado intenta adivinar un nombre mediante la creación de una `String` de la misma longitud de forma completamente aleatoria.

Si observa primero el bucle interno, el código inicializa `char asciiChar` en una letra minúscula de forma aleatoria. A continuación, estos valores `chars` se agregan a `String guess`, hasta que `String` tiene la misma longitud que el valor de `String` con el que se está comparando. Observe la comodidad del operador de concatenación aquí, que permite la concatenación de `String` y `char`.

El bucle externo prueba si la suposición es igual que la versión en minúsculas del nombre original.

Si no es así, `guess` se restablece en una `String` vacía y se vuelve a ejecutar el bucle interno, normalmente millones de veces para un nombre de cinco letras. (Tenga en cuenta que los nombres con más de cinco letras tardarán mucho tiempo).

Temas

- Creación de un bucle `while`
- Desarrollo de un bucle `for`
- Anidamiento de un bucle `for` y un bucle `while`
- **Uso de una matriz en un bucle `for`**
- Codificación y anidamiento de un bucle `do/while`
- Comparación de construcciones de bucle

ORACLE

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

Bucles y matrices

Uno de los usos más comunes de los bucles es el de trabajar con juegos de datos.

Todos los tipos de bucles resultan útiles:

- Bucles `while` (para comprobar un valor concreto)
- Bucles `for` (para pasar por toda la matriz)
- Bucles `for` mejorados

ORACLE

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

Bucle for con matrices

ages (matriz de tipos int)

27	12	82	...	1
----	----	----	-----	---

El índice empieza en 0.

El último índice de la matriz es `ages.length - 1`.

`ages[i]` accede a valores de matriz conforme `i` pasa de 0 a `ages.length - 1`.

```
for (int i = 0; i < ages.length; i++ ) {  
    System.out.println("Age is " + ages[i] );  
}
```

ORACLE

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

Salida:

Age is 27

Age is 12

Age is 82

...

Age is 1

Definición de valores en una matriz

ages (matriz de tipos int)



El bucle accede a cada elemento de la matriz por turnos.

```
for (int i = 0; i < ages.length; i++ ) {  
    ages[i] = 10;  
}
```

Cada elemento de la matriz está definido en 10.

ORACLE

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

Bucle for mejorado con matrices

ages (matriz de tipos int)



El bucle accede a cada elemento de la matriz por turnos.

Cada iteración devuelve el siguiente elemento de la matriz en age.

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

ORACLE

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

Bucle for mejorado con ArrayLists

names (ArrayList de tipos String)

George

Jill

Xinyi

...

Ravi

El bucle accede a cada elemento de la ArrayList por turnos.

Cada iteración devuelve el siguiente elemento de la ArrayList en name.

```
for (String name : names ) {  
    System.out.println("Name is " + name);  
}
```

ORACLE

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

Las ArrayLists se pueden iterar exactamente de la misma forma que las matrices.

Uso de break con bucles

Ejemplo de break:

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

No es necesario volver a pasar por el bucle, por lo tanto utilizar break.

Salida:

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

ORACLE

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

Hay dos palabras clave útiles que se pueden utilizar al trabajar con bucles: `break` y `continue`. `break` le permite salir de un bucle, mientras que `continue` le devuelve al principio del bucle.

En el ejemplo de la diapositiva se muestra el uso de `break`. Suponiendo que el código es para averiguar si alguna de las puntuaciones de la matriz es superior a `passmark`, puede definir `passed` en `true` y salir del bucle cuando se encuentre la primera puntuación de este tipo.

Uso de `continue` con bucles

Ejemplo de `continue`:

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

Si la unidad falla, seguir comprobando la siguiente unidad.

ORACLE

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

En el ejemplo de la diapositiva se muestra el uso de `continue` en un ejemplo similar. En este caso, supongamos que desea saber si se ha alcanzado un determinado número de pasadas. Por lo tanto, el enfoque es comprobar en primer lugar si la puntuación de la unidad no es suficiente. Si es así, el comando `continue` vuelve al principio del bucle. Si la puntuación es suficiente, se disminuye el número de `passesReqd` y posiblemente se realiza un procesamiento posterior.

Este ejemplo y en el anterior solo sirven para mostrar cuáles son las funciones de `break` y `continue` y no para mostrar técnicas de programación concretas. Ambos tienen una función similar: garantizan que partes del bucle no se procesen de forma innecesaria. A veces esto también se logra mediante el diseño de bloques `if`, pero resulta útil disponer de estas dos opciones en algoritmos complejos.

Temas

- Creación de un bucle `while`
- Desarrollo de un bucle `for`
- Anidamiento de un bucle `for` y un bucle `while`
- Uso de una matriz en un bucle `for`
- **Codificación y anidamiento de un bucle `do/while`**
- Comparación de construcciones de bucle

ORACLE

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

Codificación de un bucle `do/while`

Sintaxis:

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

ORACLE

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

El bucle `do/while` es un bucle iterativo de uno a varios: la condición está al final del bucle y se procesa *después del cuerpo*. El *cuerpo del bucle* se procesa, por lo tanto, al menos una vez. Si desea que la sentencia o las sentencias del cuerpo se procesen al menos una vez, utilice un bucle `do/while` en lugar de un bucle `while` o `for`. En la diapositiva se muestra la sintaxis del bucle `do/while`.

Codificación de un bucle do/while

```
setFloor() {  
    // Normally you would pass the desiredFloor as an argument to the  
    // setFloor method. However, because you have not learned how to  
    // do this yet, desiredFloor is set to a specific number (5)  
    // below.  
    int desiredFloor = 5;  
  
    do {  
        if (currentFloor < desiredFloor) {  
            goUp();  
        }  
        else if (currentFloor > desiredFloor) {  
            goDown();  
        }  
    }  
    while (currentFloor != desiredFloor);  
}
```

ORACLE

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

El método `setFloor` de la clase `Elevator` utiliza un bucle `do/while` para determinar si el ascensor está en la planta deseada. Si el valor de la variable `currentFloor` no es igual al valor de la variable `desiredFloor`, el ascensor sigue subiendo o bajando.

Temas

- Creación de un bucle `while`
- Desarrollo de un bucle `for`
- Anidamiento de un bucle `for` y un bucle `while`
- Uso de una matriz en un bucle `for`
- Codificación y anidamiento de un bucle `do/while`
- Comparación de construcciones de bucle

ORACLE

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

Comparación de construcciones de bucle

- Utilice el bucle `while` para iterar indefinidamente con las sentencias y para ejecutar las sentencias cero o más veces.
- Utilice el bucle `do/while` para iterar indefinidamente con las sentencias y para ejecutar las sentencias *una* o más veces.
- Utilice el bucle `for` para pasar por las sentencias un número predefinido de veces.

ORACLE

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

Prueba

_____ permiten comprobar y volver a comprobar una decisión para ejecutar o volver a ejecutar un bloque de código.

- a. Las clases
- b. Los objetos
- c. Los bucles
- d. Los métodos

ORACLE

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

Respuesta: c

Prueba

¿Cuál de los siguientes bucles siempre se ejecuta al menos una vez?

- a. El bucle `while`
- b. El bucle `while` anidado
- c. El bucle `do/while`
- d. El bucle `for`

ORACLE

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

Respuesta: c

Resumen

En esta lección, debe haber aprendido lo siguiente:

- Crear un bucle `while`
- Anidar un bucle `while`
- Desarrollar y anidar un bucle `for`
- Codificar y anidar un bucle `do/while`
- Utilizar una `ArrayList` en un bucle `for`
- Comparar construcciones de bucle

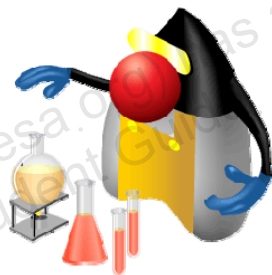


ORACLE

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

Visión general de la práctica 9-1: Escritura de una clase que utiliza un bucle `for`

En esta práctica, creará la clase `Counter` que utiliza un bucle `for` simple para imprimir una secuencia de números.



ORACLE

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

Visión general de la práctica 9-2: Escritura de una clase que utiliza un bucle `while`

En esta práctica, escribirá una clase denominada `Sequence` que muestre una secuencia que empiece con los números 0 y 1. Los siguientes números de la secuencia son la suma de los dos anteriores (por ejemplo, 0 1 1 2 3 5 8 13 21...). Esta secuencia también se denomina sucesión de Fibonacci.

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.

Visión general de la práctica de comprobación 9-3: Conversión de un bucle `while` en un bucle `for`

En esta práctica, convertirá un bucle `while` existente en un bucle `for`. En esta práctica, podrá:

- Crear una nueva clase, `ChallengeSequence`, basada en la clase `Sequence` creada en la práctica anterior.
- Modificar el método `displaySequence` para utilizar un bucle `for` en lugar de un bucle `while`.

Nota: esta práctica (9-3) es una práctica de comprobación opcional.

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.

Visión general de la práctica 9-4: Uso de bucles `for` para procesar una `ArrayList`

En esta práctica, creará dos nuevos métodos en dos clases diferentes. Esta práctica consta de dos secciones:

- Uso de un bucle `for` con la clase `VacationScaleTwo`
- Uso de un bucle `for` mejorado con la clase `NamesListTwo`

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.

Visión general de la práctica 9-5: Escritura de una clase que utiliza un bucle `for` anidado para procesar una matriz bidimensional

En esta práctica, creará y procesará una matriz bidimensional mediante un bucle `for` anidado.

ORACLE

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

Visión general de la práctica de comprobación 9-6: Adición de un método de búsqueda a `ClassMap`

En esta práctica, agregará otro método a `ClassMap`. Este método busca en `deskArray` un nombre concreto.

Nota: esta práctica (9-6) es una práctica de comprobación opcional.

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.

10

Trabajar con métodos y sobrecarga de métodos

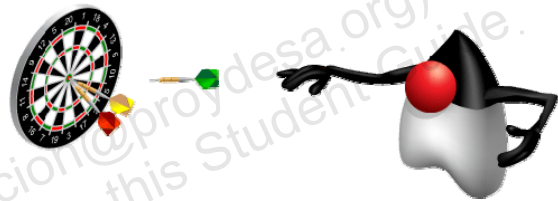
ORACLE®

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

Objetivos

Al finalizar esta lección, debería estar capacitado para:

- Declarar métodos con argumentos y valores de retorno
- Declarar métodos estáticos y variables estáticas
- Crear un método sobrecargado



ORACLE

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

Temas

- Creación y llamada a métodos
- Métodos estáticos y variables estáticas
- Sobrecarga de métodos

ORACLE

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

Creación y llamada a métodos

Sintaxis:

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

ORACLE

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

Forma básica de un método

La palabra clave `void` indica que el método no devuelve un valor.

Los paréntesis vacíos indican que no se ha transferido ningún argumento al método.

```
public void display() {  
    System.out.println("Shirt ID: " + shirtID);  
    System.out.println("Shirt description:" + description);  
    System.out.println("Color Code: " + colorCode);  
    System.out.println("Shirt price: " + price);  
} // end of display method
```

ORACLE

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

Éste es un ejemplo de un método simple que no recibe ningún argumento ni devuelve un valor.

Llamada a un método en una clase diferente

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

Salida:

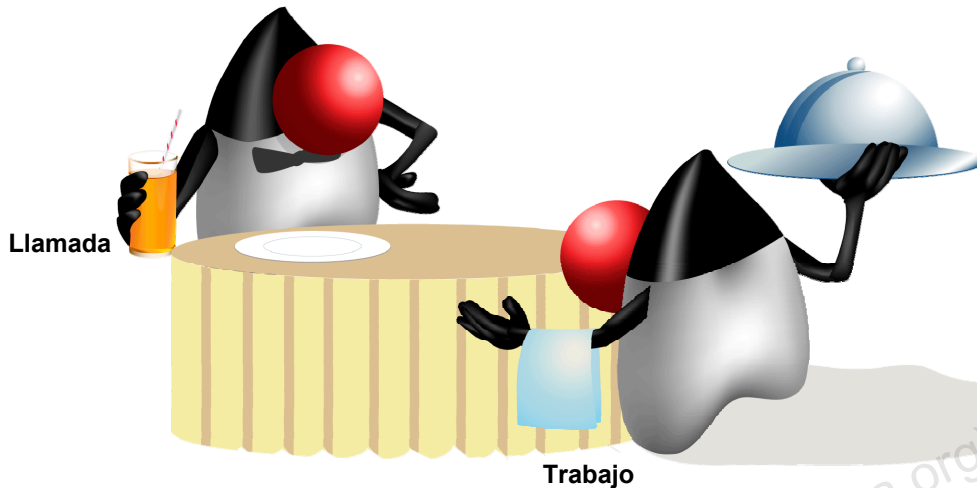
```
Item ID: 0  
Item description:-description required-  
Color Code: U  
Item price: 0.0
```

ORACLE

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

En el ejemplo de la diapositiva, se llama a `display()`. Debido a que el objeto `Shirt` no tiene ninguno de sus campos definidos, se muestran los valores por defecto de esos campos.

Métodos de llamada y de trabajo



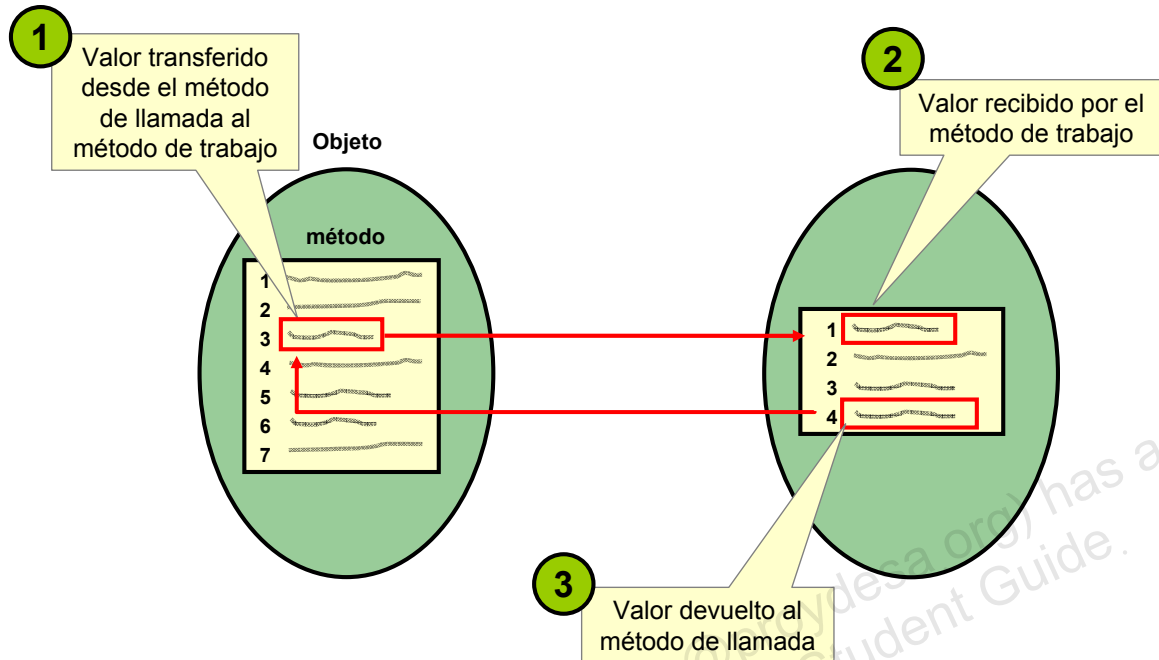
ORACLE

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

En el ejemplo anterior, la clase `ShirtTest` llama al método `display()` desde otro método (método `main`). Por lo tanto, se hace referencia al método `main` como el *método de llamada* porque “llama” a otro método para que realice algún trabajo. Por el contrario, se hace referencia al método `display` como el *método de trabajo* porque realiza determinado trabajo para el método `main`.

Cuando un método de llamada llama a un método de trabajo, el método de llamada deja de ejecutarse hasta que termina el método de trabajo. Una vez terminado el método de trabajo, el flujo de programa vuelve al punto posterior a la llamada al método en el método de llamada.

Transferencia de argumentos y devolución de valores

**ORACLE**

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

Creación de un método con un parámetro

Llamada:

```
Elevator theElevator = new Elevator();

theElevator.setFloor(4); // Send elevator to the fourth floor
```

Llamada al método `setFloor()`, transfiriendo el valor 4, de tipo `int`.

Trabajo:

```
public void setFloor(int desiredFloor) {
    while (currentFloor != desiredFloor) {
        if (currentFloor < desiredFloor) {
            goUp();
        }
        else {
            goDown();
        }
    }
}
```

El método `setFloor()` recibe un argumento de tipo `int` y le asigna el nombre `desiredFloor`.

ORACLE

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

El ejemplo de la diapositiva muestra el método `setFloor()` (que se describe en la lección “Uso de construcciones de bucle”). El método recibe un valor de tipo `int` y le asigna el nombre `desiredFloor`. `desiredFloor` es ahora una variable local cuyo ámbito es el método.

Se llama al método (en este caso, desde un método de llamada de otra clase) mediante el uso de la notación de puntos y la inclusión del argumento.

Nota: una variable definida en la declaración del método se denomina *parámetro del método*, mientras que un valor transferido a la llamada al método se denomina *argumento*.

Creación de un método con un valor de retorno

Llamada:

... < lines of code omitted > ...

```
boolean isOpen = theElevator.checkDoorStatus() // Is door open?
```

La variable local `isOpen` indica si la puerta del ascensor está abierta.

Trabajo:

```
public class Elevator {
    public boolean doorOpen=false;
    public int currentFloor = 1;
```

... < lines of code omitted > ...

```
public boolean checkDoorStatus() {
```

```
    return doorOpen;
```

```
}
```

El valor `Elevator` tiene el campo `doorOpen` para indicar el estado de la puerta del ascensor.

El tipo devuelto por el método se define después del nombre del método.

La sentencia `return` devuelve el valor en `doorOpen`.

ORACLE

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

El ejemplo de la diapositiva muestra el método `checkDoorStatus()` al que está llamando el método de llamada. Observe cómo `checkDoorStatus()` define que se devolverá un valor booleano. Aquí se puede definir cualquier tipo único; de lo contrario, se utilizará la palabra clave `void` si el método no devuelve un valor.

La sentencia `return` devuelve el valor a la sentencia de llamada. Observe que, debido a que el método se ha declarado con un tipo de retorno `boolean`, NetBeans indica un error si no hay retorno o si el retorno es de un tipo incorrecto.

Llamada a un método en la misma clase

```
public class Elevator {  
  
    public boolean doorOpen=false;  
    public int currentFloor = 1;  
  
    public final int TOP_FLOOR = 5;  
    public final int BOTTOM_FLOOR = 1;  
  
    public void openDoor() {  
  
        // Check if door already open  
        if ( !checkDoorStatus() ) {  
  
            // door opening code  
        }  
    }  
}
```

Se evalúa como true si la puerta está cerrada.

ORACLE

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

Llamar a un método en la misma clase resulta muy sencillo. Simplemente utilice el nombre del método sin una notación de puntos ni referencia. Lo mismo ocurre cuando se accede a un campo, simplemente se utiliza el nombre del campo.

Sin embargo, si tiene variables locales con nombres similares y desea que resulte obvio que el código está accediendo a un campo o método del objeto actual, puede utilizar la palabra clave `this` con la notación de puntos. `this` es una referencia al objeto actual.

Ejemplo:

```
this.checkDoorStatus()
```

Transferencia de argumentos a métodos

```
public class ShirtTest {
    public static void main (String[] args) {
        Shirt myShirt = new Shirt();
        System.out.println("Shirt color: " + myShirt.colorCode);
        changeShirtColor(myShirt, 'B');
        System.out.println("Shirt color: " + myShirt.colorCode);
    }
    public static void changeShirtColor(Shirt theShirt, char color) {
        theShirt.colorCode = color;
    }
}
```

theShirt es una
nueva referencia de
tipo Shirt.

Salida:

```
Shirt color: U
Shirt color: B
```

ORACLE

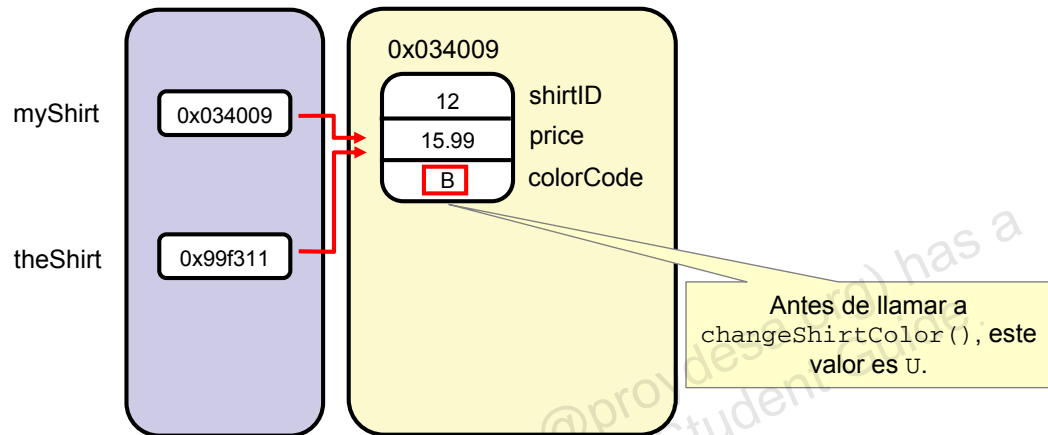
Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Al llamar al método, los valores de los argumentos se utilizan para inicializar variables de parámetro recién creadas, cada una del tipo declarado, antes de ejecutar el cuerpo del método o constructor. Esto es válido para los tipos primitivos y los tipos de referencia. (Los objetos no se transfieren a los métodos).

Esto significa que, en el ejemplo de la diapositiva, la referencia `myShirt` se transfiere mediante el valor al método `changeShirtColor()`. La referencia `theShirt` incluida en el método es una referencia diferente a `myShirt`. Sin embargo, ambas apuntan al mismo objeto, por lo que el cambio de color realizado con `theShirt` se imprime al acceder a `myShirt.color`.

Transferencia por valor

```
Shirt myShirt = new Shirt();  
changeShirtColor(myShirt, 'B');
```



ORACLE

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

El diagrama de la diapositiva muestra cómo el valor de la referencia `myShirt` transferido al método `changeShirtColor()` se utiliza para inicializar una nueva referencia `Shirt` (en este caso, `theShirt`).

Transferencia por valor

```
public class ShirtTest {  
    public static void main (String[] args) {  
        Shirt myShirt = new Shirt();  
        System.out.println("Shirt color: " + myShirt.colorCode);  
        changeShirtColor(myShirt, 'B');  
        System.out.println("Shirt color: " + myShirt.colorCode);  
    }  
    public static void changeShirtColor(Shirt theShirt, char color) {  
        theShirt = new Shirt();  
        theShirt.colorCode = color;  
    }  
}
```

Salida:

```
Shirt color: U  
Shirt color: U
```

ORACLE

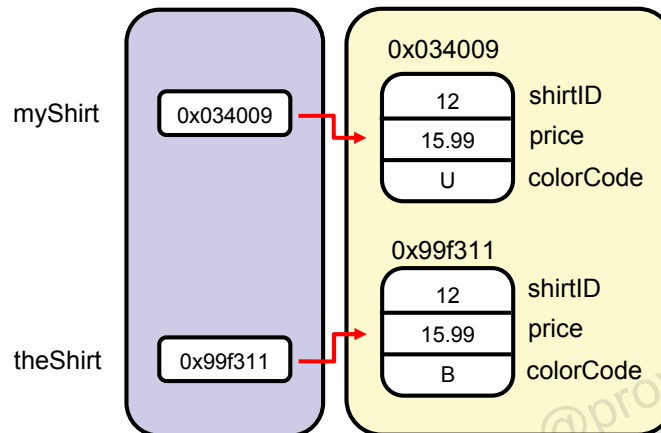
Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Aquí se muestra otro ejemplo con un pequeño cambio en el código del método `changeShirtColor()`. En este ejemplo, el valor de referencia transferido al método se asigna al valor `new shirt`. A continuación, como ha ocurrido antes, el color del objeto `Shirt` se cambia a 'B'. Sin embargo, en este caso, la línea impresa después de la llamada al método muestra que el color aún es 'U' (no definido).

Este ejemplo ilustra que la referencia `myShirt`, efectivamente, se ha transferido por valor. Los cambios realizados a las referencias transferidas a los métodos de trabajo no afectan a las referencias del método de llamada. (Observe que esta información está relacionada con los cambios realizados a las referencias transferidas al método, no a los objetos a los que apuntan).

Transferencia por valor

```
Shirt myShirt = new Shirt();
changeShirtColor(myShirt, 'B');
```



ORACLE

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

El diagrama de la diapositiva muestra la situación resultante del código de la diapositiva anterior. Al transferir `myShirt` al método `changeShirtColor()`, se inicializa una nueva variable de referencia, `theShirt` con el valor `myShirt`. Inicialmente, esta referencia apunta al objeto al que apunta la referencia `myShirt`. Sin embargo, después de asignar un nuevo objeto `Shirt` a `theShirt`, cualquier cambio realizado con `theShirt` sólo afectará al nuevo objeto `Shirt`.

Ventajas del uso de métodos

Los métodos:

- Facilitan la lectura y el mantenimiento de los programas
- Aceleran el desarrollo y el mantenimiento
- Son básicos para el software reutilizable
- Permiten la comunicación entre objetos independientes y la distribución del trabajo realizado por el programa

ORACLE

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

Prueba

¿Cuáles de las siguientes afirmaciones son ciertas?
(Seleccione todas las respuestas posibles).

- a. Una clase sólo puede contener una declaración del método.
- b. Un método siempre debe especificar un tipo de retorno.
- c. El mismo método puede ser tanto un método de trabajo como un método de llamada.
- d. No es necesario mostrar los argumentos en el mismo orden en la llamada al método que en la firma de método.

ORACLE

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

Respuesta: b, c

Métodos de llamada: Resumen

- No hay ningún límite establecido para el número de llamadas al método que puede realizar un método de llamada.
- El método de llamada y el método de trabajo pueden estar en la misma clase o en clases diferentes.
- La forma en la que se llama al método de trabajo varía en función de si está en la misma clase o en una clase diferente a la del método de llamada.
- Puede llamar a los métodos en cualquier orden.
 - No es necesario completar los métodos en el orden en el que aparecen en la clase en la que están declarados (la clase que contiene los métodos de trabajo).
- Todos los argumentos transferidos a un método se transfieren por valor.

ORACLE

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

Temas

- Creación y llamada a métodos
- **Métodos estáticos y variables estáticas**
- Sobrecarga de métodos

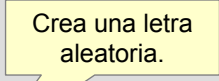
ORACLE®

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

Utilidades matemáticas

```
String name = "Lenny";
String guess = "";
int numTries = 0;

while (!guess.equals(name.toLowerCase())) {
    guess = "";
    while (guess.length() < name.length()) {
        char asciiChar = (char) (Math.random() * 26 + 97);
        guess = guess + asciiChar;
    }
    numTries++;
}
System.out.println(name + " found after " + numTries + " tries!");
```

**ORACLE**

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

Esta diapositiva vuelve a acceder al código utilizado en la lección “Uso de construcciones de bucle”, pero una parte (la parte en la que se genera la letra) no se ha explicado en esa lección.

Los valores de caracteres ASCII codifican las letras en minúsculas de la a a la z, desde 97 a 122. Al generar un número que está dentro de ese rango y colocarlo en un elemento `char`, puede utilizar el operador de concatenación para crear una cadena, tal y como se muestra aquí.

Nota: Java realmente utiliza Unicode, no ASCII, pero los primeros 128 caracteres de Unicode y ASCII son los mismos.

En la siguiente diapositiva, observe con más atención el método `Math.random()` y observe de qué tipo de método se trata.

Métodos estáticos de Math

Observe que el tipo es `double` y que es estático.

Éste es el método aleatorio.

<code>static double</code>	<code>pow(double a, double b)</code> Returns the value of the first argument raised to the power of the second argument.
<code>static double</code>	<code>random()</code> Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.
<code>static double</code>	<code>rint(double a)</code> Returns the double value that is closest in value to the argument and is equal to a mathematical integer.
<code>static long</code>	<code>round(double a)</code> Returns the closest long to the argument, with ties rounding up.

ORACLE

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

La captura de pantalla muestra una pequeña selección de métodos de la clase `Math`. El método que observamos aquí es `random()`. Devuelve un valor `double` entre 0 y 1. Para generar un valor `double` entre 0 y 10, sólo tiene que multiplicar por 10:

```
Math.random * 10
```

O bien, para generar un valor `double` entre 1 y 10, multiplique por 9 y agregue 1.

Puede que desee un valor `integer` en lugar de `double`, por lo que necesitará convertirlo en `int` o, en el caso del ejemplo de la página anterior, en `char`.

Observe que el método es estático y, por lo tanto, lo son todos los métodos en `Math`. Esto significa que `Math` no se necesita instanciar para llamar a ninguno de sus métodos (de hecho, `Math` no se puede instanciar).

Puede llamar a los métodos estáticos de una clase con la siguiente sintaxis:

```
<classname>.<method_name>
```

Creación de métodos y variables `static`

Los métodos y las variables no locales pueden ser estáticos.

- Pertenecen a la clase, no al objeto.
- Se declaran con la palabra clave `static`:

```
static Properties getProperties()
```

- Para llamar a los métodos `static`:

```
Classname.method();
```

- Para acceder a las variables `static` en otra clase:

```
Classname.attribute_name;
```

- Para acceder a las variables `static` en la misma clase:

```
attribute_name;
```

ORACLE

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

Hasta ahora ha aprendido a acceder a los métodos y las variables mediante la creación de un objeto de la clase a la que pertenece el método o la variable y la llamada al método o el acceso a la variable (si se trata de una variable pública). Los métodos y las variables que son únicos en una instancia se denominan *métodos de instancia* y *variables de instancia*.

También ha utilizado métodos que no necesitan instanciación de objetos, como el método `main`. Se denominan *métodos de clase* o *métodos estáticos* y puede llamarlos sin haber creado primero un objeto.

Del mismo modo, el lenguaje de programación Java permite crear variables estáticas o variables de clase, las cuales se pueden utilizar sin crear un objeto.

Creación de métodos y variables `static`

```
public static char convertShirtSize(int numericalSize) {  
    if (numericalSize < 10) {  
        return 'S';  
    }  
    else if (numericalSize < 14) {  
        return 'M';  
    }  
  
    else if (numericalSize < 18) {  
        return 'L';  
    }  
  
    else {  
        return 'X';  
    }  
}
```

ORACLE

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

La diapositiva muestra un ejemplo de un método que se puede agregar a la clase `Shirt` para convertir los tamaños numéricos de camisa a tamaños como pequeña, mediana y grande. Este método es estático porque:

- No utiliza directamente ningún atributo de la clase `Shirt`.
- Puede que desee llamar al método incluso si no tiene un objeto `Shirt`.

El método `convertShirtSize` acepta un tamaño numérico, determina el tamaño de carácter correspondiente (S, M, L o X) y devuelve el tamaño del carácter.

Por ejemplo, para acceder al método estático `convertShirtSize()` de la clase `Shirt`:

```
char size = Shirt.convertShirtSize(16);
```

Variables static

- Declaración de variables `static`:

```
static double salesTAX = 8.25;
```

- Acceso a variables `static`:

```
Classname.variable;
```

- Ejemplo:

```
double myPI;
myPI = Math.PI;
```



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

También puede utilizar la palabra clave `static` para declarar una variable de clase. Esto significa que sólo puede existir una copia de la variable en la memoria asociada a una clase en lugar de una copia para cada instancia de objeto.

En el ejemplo de la diapositiva, `salesTAX` es una variable estática. Puede acceder a esta variable desde cualquier método en cualquier clase mediante el nombre de clase de su clase. Suponga que se encuentra en una clase denominada `TaxUtilities`. Podría acceder a ella mediante el siguiente código:

```
TaxUtilities.salesTAX
```

O, si `TaxUtilities` tiene métodos, éstos (estáticos o de instancia) pueden acceder a la variable por nombre, sin el nombre de clase:

```
salesTAX
```

Observe que las variables pueden tener el modificador estático y final para indicar que sólo existe una copia de la variable y que su contenido no se puede cambiar. La variable `PI` de la clase `Math` es una variable final estática.

Métodos estáticos y variables estáticas en la API de Java

Ejemplos

- Algunas de las funcionalidades de la clase `Math` son:
 - Exponencial
 - Logarítmica
 - Trigonométrica
 - Aleatoria
 - Acceso a las constantes matemáticas comunes, como el valor pi (`Math.PI`)
- Algunas de las funcionalidades de la clase `System` son:
 - Recuperación de variables de entorno
 - Acceso a los flujos de entrada y salida estándar
 - Salida del programa actual (`System.exit()`)

ORACLE

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

Algunas bibliotecas de clases Java, como la clase `System` y `Math`, sólo contienen métodos estáticos y variables estáticas. La clase `System` contiene métodos de utilidad para manejar tareas específicas del sistema operativo. (No funcionan en una instancia de objeto). Por ejemplo, el método `getProperties` de la clase `System` obtiene información sobre la computadora que está utilizando.

La clase `Math` contiene métodos de utilidad para operaciones matemáticas.

Métodos estáticos y variables estáticas en la API de Java

Para declarar una variable o método `static` se debe tener en cuenta que:

- La realización de la operación en un objeto individual o la asociación de la variable a un tipo de objeto específico no es importante.
- El acceso a la variable o el método antes de instanciar un objeto sí es importante.
- El método o la variable no pertenece lógicamente a un objeto, pero posiblemente pertenezca a una clase de utilidad, como la clase `Math`, incluida en la API de Java.

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.


Temas

- Creación y llamada a métodos
- Métodos estáticos y variables estáticas
- Sobrecarga de métodos

ORACLE

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

Firma de método



```
public int getYearsToDouble(int initialSum, int interest) {  
    int interest = 7;           // per cent  
    int years = 0;  
    int currentSum = initialSum * 100; // Convert to pennies  
    int desiredSum = currentSum * 2;  
    while ( currentSum <= desiredSum) {  
        currentSum += currentSum * interest/100;  
        years++;  
    }  
}
```

ORACLE

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

El ejemplo muestra algún código de la lección en bucles, reescrito como un método que tiene dos parámetros (la suma inicial de dinero y el tipo de interés) y devuelve el número de años necesarios para duplicar la suma inicial.

La llamada muestra la parte de la declaración del método denominada *firma de método*.

La firma de método de un método es la combinación única del nombre del método y el número, los tipos y el orden de sus parámetros. La firma de método no incluye el tipo de retorno.

Sobrecarga de métodos

Métodos sobrecargados:

- Tienen el mismo nombre.
- Tienen firmas diferentes.
 - Número, tipo u orden de los parámetros diferente.
- Pueden tener funcionalidades diferentes o similares.
- Se utilizan ampliamente en las clases base.

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.

En el lenguaje de programación Java, una clase puede incluir varios métodos que tienen el mismo nombre pero argumentos diferentes (por lo que la firma de método es diferente). Este concepto se denomina *sobrecarga de métodos*. Al igual que puede distinguir entre dos estudiantes de la misma clase que se llaman “Jim” llamándolos “Jim el de la camisa verde” y “Jim el del localizador”, puede distinguir entre dos métodos por el nombre y los argumentos.

Uso de la sobrecarga de métodos

```
public final class Calculator {  
  
    public static int sum(int numberOne, int numberTwo){  
        System.out.println("Method One");  
        return numberOne + numberTwo;  
    }  
  
    public static float sum(float numberOne, float numberTwo) {  
        System.out.println("Method Two");  
        return numberOne + numberTwo;  
    }  
  
    public static float sum(int numberOne, float numberTwo) {  
        System.out.println("Method Three");  
        return numberOne + numberTwo;  
    }  
}
```

ORACLE

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

El ejemplo de la diapositiva muestra tres métodos para sumar dos números, como dos tipos `int` o dos tipos `float`. Con la sobrecarga de métodos, puede crear varios métodos con el mismo nombre y diferentes firmas.

El primer método `sum` acepta dos argumentos `int` y devuelve un valor `int`. El segundo método `sum` acepta dos argumentos `float` y devuelve un valor `float`. El tercer método `sum` acepta `int` y `float` como argumentos y devuelve un valor `float`.

Para llamar a cualquiera de los métodos `sum`, el compilador compara la firma de método de la llamada al método con las firmas de método de una clase.

Uso de la sobrecarga de métodos

```
public class CalculatorTest {  
  
    public static void main(String [] args) {  
  
        int totalOne = Calculator.sum(2,3);  
        System.out.println("The total is " + totalOne);  
  
        float totalTwo = Calculator.sum(15.99F, 12.85F);  
        System.out.println(totalTwo);  
  
        float totalThree = Calculator.sum(2, 12.85F);  
        System.out.println(totalThree);  
    }  
}
```

ORACLE

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

El ejemplo de código de la diapositiva tiene un método `main` que llama a cada uno de los métodos `sum` anteriores de la clase `Calculator`.

Sobrecarga de métodos y la API de Java

Método	Uso
<code>void println()</code>	Termina la línea actual escribiendo la cadena de separador de líneas.
<code>void println(boolean x)</code>	Imprime un valor booleano y, a continuación, termina la línea.
<code>void println(char x)</code>	Imprime un carácter y, a continuación, termina la línea.
<code>void println(char[] x)</code>	Imprime una matriz de caracteres y, a continuación, termina la línea.
<code>void println(double x)</code>	Imprime un valor <code>double</code> y, a continuación, termina la línea.
<code>void println(float x)</code>	Imprime un valor <code>float</code> y, a continuación, termina la línea.
<code>void println(int x)</code>	Imprime un valor <code>int</code> y, a continuación, termina la línea.
<code>void println(long x)</code>	Imprime un valor <code>long</code> y, a continuación, termina la línea.
<code>void println(Object x)</code>	Imprime un objeto y, a continuación, termina la línea.
<code>void println(String x)</code>	Imprime una cadena y, a continuación, termina la línea.

ORACLE

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

Varios métodos de la API de Java están sobrecargados, incluido el método `System.out.println`. La tabla de la diapositiva muestra todas las variaciones del método `println`.

Prueba

¿Qué método corresponde a la siguiente llamada al método?

```
myPerson.printValues(100, 147.7F, "lavender");
```

- a. `public void printValues (int pantSize, float ageInYears)`
- b. `public void printValues (pantSize, float ageInYears, favoriteColor)`
- c. `public void printValues (int pantSize, float ageInYears, String favoriteColor)`
- d. `public void printValues (float ageInYears, String favoriteColor, int pantSize)`

ORACLE

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

Respuesta: c

Resumen

En esta lección, debe haber aprendido lo siguiente:

- Declarar métodos con argumentos y valores de retorno
- Declarar métodos estáticos y variables estáticas
- Crear un método sobrecargado

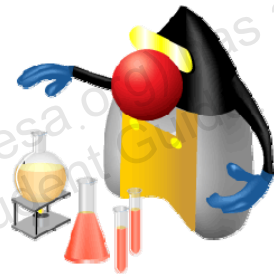


ORACLE

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

Visión general de la práctica 10-1: Escritura de un método con argumentos y valores de retorno

En esta práctica, creará una clase para realizar un pedido de más de una camisa y, a continuación, mostrará el valor total del pedido de camisas.



ORACLE

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

Visión general de la práctica de comprobación 10-2: Escritura de una clase que contenga un método sobrecargado

En esta práctica, escribirá una clase `Customer` con un método sobrecargado denominado `setCustomerInfo()`.

Nota: esta práctica (10-2) es una práctica de comprobación opcional.

ORACLE

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

11

Uso de encapsulación y constructores

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:

- Utilizar modificadores de acceso
- Describir el objetivo de la encapsulación
- Implantar la encapsulación en una clase
- Crear un constructor



ORACLE

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

Temas

- Encapsulación
- Constructores

ORACLE

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

Visión general

- Encapsulación significa ocultar los campos de objeto mediante la conversión de todos los campos en privados:
 - Utilizar los métodos getter y setter.
 - En los métodos setter, utilice el código para asegurarse de que los valores son válidos.
- La encapsulación exige una programación a la interfaz:
 - El tipo de dato del campo es irrelevante para el método de llamada.
 - La clase se puede cambiar mientras que la interfaz sea la misma.
- La encapsulación fomenta un diseño orientado a objetos (OO) correcto.

ORACLE

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

Modificador `public`

```
public class Elevator {  
    public boolean doorOpen=false;  
    public int currentFloor = 1;  
    public final int TOP_FLOOR = 10;  
    public final int MIN_FLOOR = 1;  
  
    ... < code omitted > ...  
  
    public void goUp() {  
        if (currentFloor == TOP_FLOOR) {  
            System.out.println("Cannot go up further!");  
        }  
        if (currentFloor < TOP_FLOOR) {  
            currentFloor++;  
            System.out.println("Floor: " + currentFloor);  
        }  
    }  
}
```

ORACLE

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

El código de la diapositiva muestra el método `goUp()` y el campo `currentFloor`. Se trata del método correspondiente al método `goDown()` descrito anteriormente y evita que el ascensor intente sobrepasar la planta más alta.

Pero el código que aparece aquí tiene un problema. El método `goUp()` se puede evitar; no hay ningún elemento que evite que el campo `currentFloor` se modifique directamente.

Riesgos del acceso a un campo `public`

```
Elevator theElevator = new Elevator();  
  
theElevator.currentFloor = 15; ← Puede causar un problema.
```

ORACLE

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

Modificador `private`

```
public class Elevator {  
    private boolean doorOpen=false;  
    private int currentFloor = 1;  
    private final int TOP_FLOOR = 10;  
    private final int MIN_FLOOR = 1;  
  
    ... < code omitted > ...  
  
    public void goUp() {  
        if (currentFloor == TOP_FLOOR) {  
            System.out.println("Cannot go up further!");  
        }  
        if (currentFloor < TOP_FLOOR) {  
            currentFloor++;  
            System.out.println("Floor: " + currentFloor);  
        }  
    }  
}
```

No se puede acceder a ninguno de estos campos desde otra clase con una notación de puntos.

ORACLE

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

En el ejemplo mostrado, todos los campos se han convertido en privados. Ahora no se puede acceder a ellos desde un método de llamada que no pertenezca a esta clase. Por lo tanto, cualquier método de llamada que desee controlar la planta a la que irá el ascensor debe realizar esta acción a través de sus métodos públicos.

Intento de acceso a un campo private

```
Elevator theElevator = new Elevator();  
  
theElevator.currentFloor = 15; ← No permitido
```

NetBeans
mostrará un
error. Puede
obtener una
explicación si
coloca aquí el
cursor.

```
1  public class ElevatorTest {  
2  
3  public static void main(String args[]) {  
4      currentFloorIndex has private access in Loops_Elevator  
5      --  
6      (Alt-Enter shows hints) Elevator();  
7  
8      theElevator.currentFloorIndex = 17;  
9  
10 }
```

ORACLE

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

Modificador `private` en los métodos

```
public class Elevator {
    ... < code omitted > ...

    private void setFloor() {
        int desiredFloor = 5;
        while ( currentFloor != desiredFloor ){
            if (currentFloor < desiredFloor) {
                goUp();
            } else {
                goDown();
            }
        }
    }

    public void requestFloor(int desiredFloor) {
        ... < contains code to add requested floor to a queue > ...
    }
}
```

¿Debe ser privado este método?

ORACLE

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

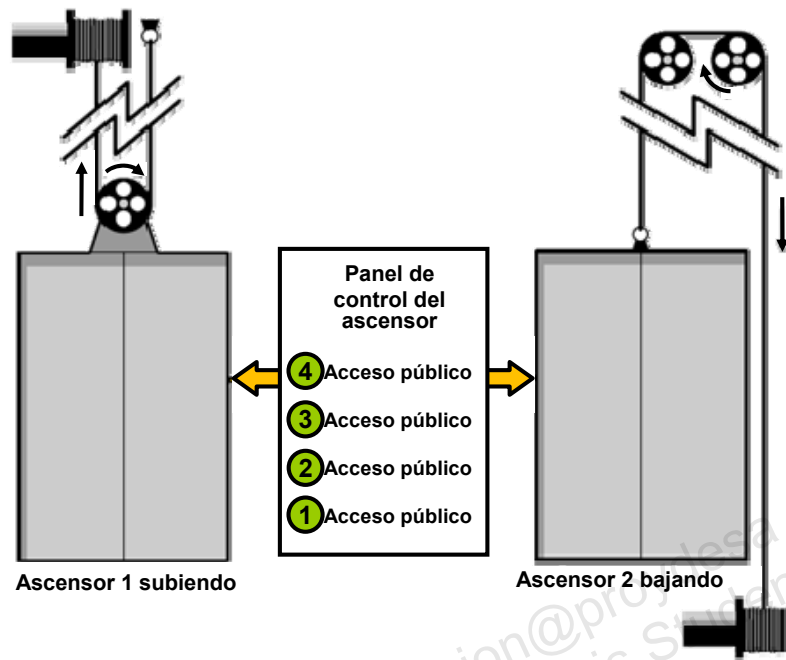
¿Recuerda el método `setFloor()`? Al igual que los campos, los métodos se declaran con un modificador. ¿Se le ocurre algún motivo por el que este método se puede declarar mejor con un modificador privado?

Si el ascensor funciona como la mayoría de ascensores, los controles que utiliza el público en general (ya sea un botón para llamar al ascensor o el botón para seleccionar una planta), no afectan directamente al ascensor.

En su lugar, un usuario presiona un botón, por ejemplo, una solicitud para que el ascensor vaya a la quinta planta. El ascensor no responde inmediatamente a la solicitud, pero la pone en cola y, finalmente, quizás después de llevar a los usuarios que ya estaban en el ascensor hasta la primera planta, irá a la quinta planta.

Puede que el único método público necesario sea `requestFloor()`, al menos para el software que controla los botones que utiliza el público en general.

Interfaz e implantación



ORACLE

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

Cuando las clases están encapsuladas, otros objetos interactúan sólo con algunas partes (métodos) de cualquier otra clase.

En el ejemplo del ascensor, el programa de control que se dispara por medio de los botones sólo puede llamar al método `requestFloor()` de `Elevator`. Mientras `Elevator` implante este método, no es importante la forma exacta en la que se implante. El método podría almacenar solicitudes en una matriz binaria en la que la definición de un elemento en `true` indicara que existe una solicitud en la planta con ese índice. También se puede utilizar una `ArrayList` para almacenar los números de plantas solicitados.

También puede que se dispare un método `moveElevator()` por algún motivo, quizás porque las puertas se están cerrando. Una vez más, mientras se implante este método `moveElevator()`, su implantación se puede variar para cambiar la forma en la que el ascensor responde a las solicitudes que se reciben al mismo tiempo desde diferentes plantas.

Métodos get y set

```
public class Shirt {  
    private int shirtID = 0; // Default ID for the shirt  
    private String description = "-description required-"; // default  
    // The color codes are R=Red, B=Blue, G=Green, U=Unset  
    private char colorCode = 'U';  
    private double price = 0.0; // Default price for all items  
  
    public char getColorCode() {  
        return colorCode;  
    }  
    public void setColorCode(char newCode) {  
        colorCode = newCode;  
    }  
    // Additional get and set methods for shirtID, description,  
    // and price would follow  
  
} // end of class
```

ORACLE

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

Si convierte los atributos en privados, ¿cómo puede acceder a ellos otro objeto? Un objeto puede acceder a los atributos privados de un segundo objeto si éste proporciona métodos públicos para cada una de las operaciones que se van a realizar en el valor de un atributo.

Por ejemplo, se recomienda que todos los campos de una clase sean privados, mientras que aquéllos a los que se necesite acceder deben tener métodos públicos para definir y obtener los valores.

Esto asegura que, en algún momento en el futuro, el tipo de campo real se pueda cambiar, si ello tuviera alguna ventaja. También podría ocurrir que los métodos getter y setter se modificaran para controlar el cambio del valor, de la misma forma que se escribió el código para garantizar que el campo `currentFloor` del ascensor no se pudiera definir en un valor no válido.

Uso de los métodos setter y getter

```
public class ShirtTest {
    public static void main (String[] args) {
        Shirt theShirt = new Shirt();
        char colorCode;

        // Set a valid colorCode
        theShirt.setColorCode('R');
        colorCode = theShirt.getColorCode();
        // The ShirtTest class can set and get a valid colorCode
        System.out.println("Color Code: " + colorCode);

        // Set an invalid color code
        theShirt.setColorCode('Z'); ← No es un código de color válido.
        colorCode = theShirt.getColorCode();
        // The ShirtTest class can set and get an invalid colorCode
        System.out.println("Color Code: " + colorCode);
    }
}
```

ORACLE

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

Aunque el código de la clase `Shirt` sea sintácticamente correcto, el método `setColorCode` no contiene ninguna lógica para asegurarse de que se definen los valores correctos.

El ejemplo de código de la diapositiva define correctamente un código de color no válido en el objeto `Shirt`.

Sin embargo, ya que `ShirtTest` accede a un campo privado en `Shirt` a través del método setter, ahora `Shirt` se puede volver a codificar sin modificar ninguna de las clases que dependen de este valor.

Método setter con comprobación

```
public void setColorCode(char newCode) {  
    switch (newCode) {  
        case 'R':  
        case 'G':  
        case 'B':  
            colorCode = newCode;  
            break;  
        default:  
            System.out.println("Invalid colorCode. Use R, G, or B");  
    }  
}
```

ORACLE

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

En la diapositiva se incluye otra versión de la clase `Shirt`. Sin embargo, en esta clase, antes de definir el valor, el método setter se asegura de que el valor es válido. Si no es válido, el campo `colorCode` permanece sin modificar y se imprime un mensaje de error.

Uso de los métodos setter y getter

```
public class ShirtTest {
    public static void main (String[] args) {
        Shirt theShirt = new Shirt();
        System.out.println("Color Code: " + theShirt.getColorCode());

        // Try to set an invalid color code
        Shirt1.setColorCode('Z'); ← No es un código de color válido.
        System.out.println("Color Code: " + theShirt.getColorCode());
    }
}
```

Salida:

```
Color Code: U ← Llamada anterior a setColorCode(): muestra un valor por defecto.
Invalid colorCode. Use R, G, or B ← La llamada a setColorCode imprime un
                                mensaje de error.
Color Code: U ← colorCode no modificado por un argumento no válido transferido a
                setColorCode()
```

ORACLE

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

Encapsulación: Resumen

La encapsulación protege los datos de la siguiente forma:

- Convierte todos los campos en privados.
 - Utiliza los métodos getter y setter.
 - En los métodos setter, utiliza el código para comprobar que los valores son válidos.
- Exige una programación a la interfaz.
 - El tipo de dato del campo es irrelevante para el método de llamada.
 - La clase se puede cambiar mientras que la interfaz sea la misma.
- Fomenta un diseño OO correcto.



ORACLE

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

Temas

- Encapsulación
- Constructores

ORACLE®

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

Inicialización de un objeto `shirt`

```
public class ShirtTest {  
    public static void main (String[] args) {  
        Shirt theShirt = new Shirt();  
  
        // Set values for the Shirt  
        theShirt.setColorCode('R');  
        theShirt.setDescription("Outdoors shirt");  
        theShirt.price(39.99);  
    }  
}
```

ORACLE

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

Suponiendo que ahora tiene métodos setters para todos los campos privados de `Shirt`, puede instanciar e inicializar un objeto `Shirt` mediante su instanciación y, a continuación, la definición de los diferentes campos a través de los métodos setter.

Sin embargo, Java proporciona una forma mucho más adecuada de instanciar e inicializar un objeto, con un método especial denominado *constructor*.

Constructores

- Los constructores son estructuras similares a un método de una clase:
 - Tienen el mismo nombre que la clase.
 - Se suelen utilizar para inicializar campos en un objeto.
 - Pueden recibir argumentos.
 - Se pueden sobrecargar.
- Todas las clases tienen al menos un constructor:
 - Si no hay constructores explícitos, el compilador Java proporciona un constructor sin argumentos por defecto.
 - Si hay más de un constructor explícito, no se proporcionará ningún constructor por defecto.

ORACLE

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

Todas las clases tienen al menos un constructor. Si el código no incluye un constructor explícito, el compilador Java proporciona automáticamente un constructor sin argumentos. A este constructor se le denomina constructor por defecto.

Creación de constructores

Sintaxis:

```
[modifiers] class ClassName {  
  
    [modifiers] ClassName([arguments]) {  
        code_block  
    }  
}
```

ORACLE

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

- `[modifiers]` representa varias palabras clave únicas de la tecnología Java que pueden modificar la forma en la que se accede a los constructores. Los modificadores son opcionales (se indican con corchetes).
- `ClassName` es el nombre de la clase y el nombre del método del constructor. El nombre del constructor debe ser el mismo que el de `ClassName` en la declaración de clase.
- `[arguments]` representa uno o varios argumentos opcionales transferidos al constructor.
- `code_block` representa una o varias líneas de código opcionales para el constructor.

Creación de constructores

```
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  
    private char colorCode = 'U';  
    public double price = 0.0; // Default price all items  
  
    // This constructor takes one argument  
    public Shirt(char colorCode ) {  
        setColorCode(colorCode);  
    }  
}
```

ORACLE

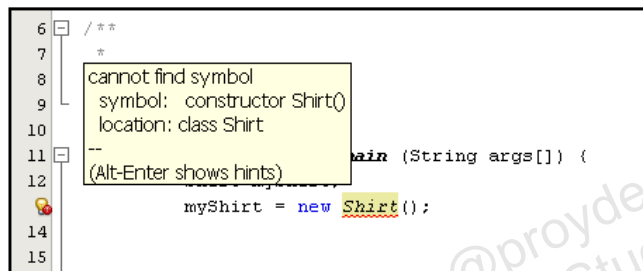
Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

El ejemplo de Shirt mostrado en la diapositiva tiene un constructor que acepta un valor `char` para inicializar el código de color para este objeto. Debido a que `setColorCode()` asegura que no se puede definir un código no válido, el constructor sólo puede llamar a este método.

Inicialización de un objeto shirt con un constructor

```
public class ShirtTest {
    public static void main (String[] args) {
        Shirt theShirt = new Shirt('G');

        theShirt.display();
    }
}
```



ORACLE

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

Como era de esperar, al transferir un código de color válido al constructor Shirt, se crea un nuevo objeto Shirt y, al llamar a `display()`, se produce la siguiente salida:

```
Item ID: 0
Item description:-description required-
Color Code: G
Item price: 0.0
```

Sin embargo, observe el mensaje que obtiene en NetBeans si intenta llamar al constructor Shirt sin ningún argumento (como ha hecho anteriormente en este curso).

El motivo del problema es que, si no hay ningún constructor explícito en una clase, Java supone que desea poder instanciar la clase y le proporciona un constructor sin argumentos por defecto. De lo contrario, ¿cómo se podría instanciar la clase?

Sin embargo, si tiene un constructor explícito, Java supone que puede que desee que éste sea el único constructor y ya no proporcionará una implantación sin argumentos por defecto.

Varios constructores

```
public class Shirt {
    ... < declarations for field omitted > ...
```

```
// No-argument constructor
public Shirt() {
    // You could add some default processing here
}
```

Si es necesario, se debe agregar explícitamente.

```
// This constructor takes one argument
public Shirt(char colorCode ) {
    setColorCode(colorCode);
}
```

```
public Shirt(char colorCode, double price) {
    this(colorCode);
    setPrice(price);
}
```

Encadenamiento de constructores.

ORACLE

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

El código de la diapositiva muestra tres constructores sobrecargados:

- Un constructor sin argumentos por defecto
- Un constructor con un parámetro (`char`)
- Un constructor con dos parámetros (`char` y `double`)

El tercer constructor define los campos `colorCode` y `price`. Sin embargo, observe que la sintaxis en la que se define el campo `colorCode` es una de las que aún no ha visto. Puede definir `colorCode` con una simple llamada a `setColorCode()`, igual que lo ha hecho el constructor anterior, pero existe otra opción, la cual se muestra a continuación.

Puede encadenar los constructores mediante la llamada al segundo constructor de la primera línea del tercer constructor con la siguiente sintaxis:

```
this(argument);
```

`this` es una palabra clave especial que es una referencia al objeto actual.

Esta técnica de encadenamiento de constructores es especialmente útil cuando un constructor tiene código (quizás algo complejo) que está asociado a la definición de campos. No desearía duplicar el código en otro constructor y, de esta forma, encadenaría los constructores.

Prueba

¿Cuál es el constructor por defecto para la siguiente clase?

```
public class Penny {  
    String name = "lane";  
}
```

- a. `public Penny(String name)`
- b. `public Penny()`
- c. `class()`
- d. `String()`

ORACLE

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

Respuesta: b

Resumen

En esta lección, debe haber aprendido lo siguiente:

- Utilizar modificadores de acceso
- Describir el objetivo de la encapsulación
- Implantar la encapsulación en una clase
- Crear un constructor



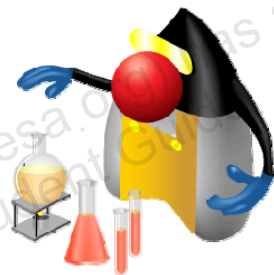
ORACLE

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

Visión general de la práctica 11-1: Implantación de la encapsulación en una clase

En esta práctica, creará una clase que contenga atributos privados e intentará acceder a ellos en otra clase. En esta práctica, podrá:

- Implantar la encapsulación en una clase
- Acceder a los atributos encapsulados de una clase

**ORACLE**

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

Visión general de la práctica de comprobación 11-2: Adición de validación a la clase `DateThree`

En esta práctica, agregará un método `setDate()` a la clase `DateThree` que realiza la validación en los valores de la parte de fecha que se transfieren al método.

Nota: esta práctica (11-2) es una práctica de comprobación opcional.

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.

Visión general de la práctica 11-3: Creación de constructores para inicializar objetos

En esta práctica, podrá crear una clase y utilizar constructores para inicializar objetos.

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.

12

Uso de conceptos orientados a objetos avanzados

ORACLE®

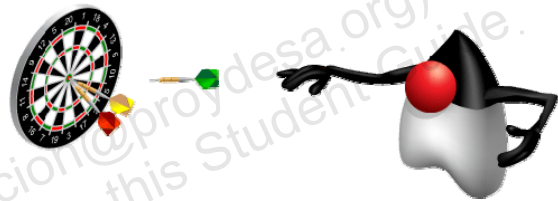
Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

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 la herencia
- Probar las relaciones de superclases y subclases
- Describir el polimorfismo
- Crear una subclase
- Utilizar interfaces y clases abstractas



ORACLE

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

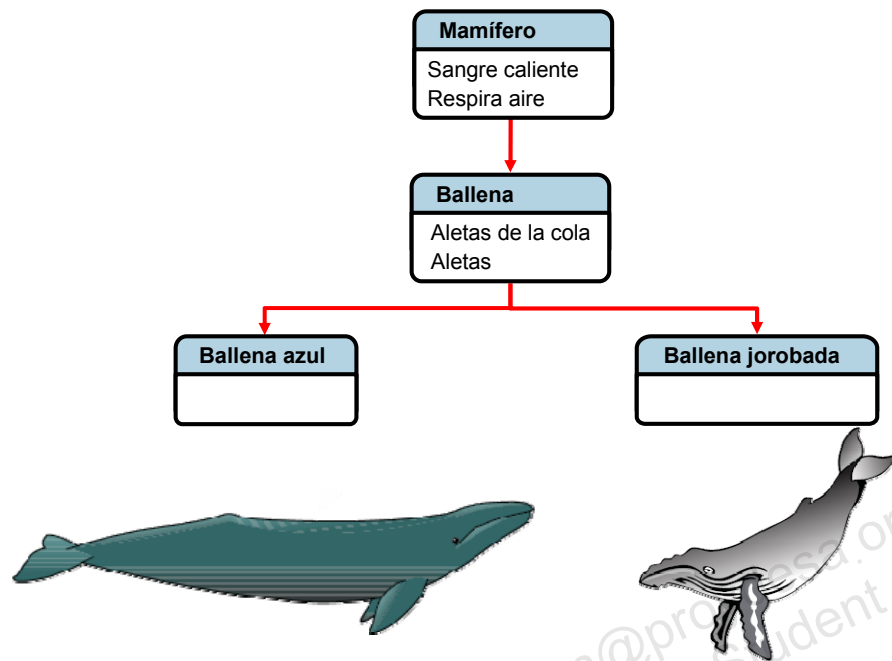
Temas

- **Visión general de la herencia**
- Trabajar con superclases y subclases
- Polimorfismo y sustitución de métodos
- Interfaces
- Clase Object

ORACLE

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

Jerarquías de clase



ORACLE

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

La herencia resulta en una jerarquía de clases de las clases de tecnología Java similar a las taxonomías de la biología, como “Ballena azul es una subclase de Ballena”.

El diagrama de la diapositiva ilustra una jerarquía de las ballenas. “Sangre caliente” es un atributo de la superclase Mamífero. La frase “Respira aire” representa una operación que también forma parte de la superclase Mamífero. Las aletas de la cola y las aletas son atributos específicos de la clase Ballena, que es una subclase de la clase Mamífero.

Temas

- Visión general de la herencia
- **Trabajar con superclases y subclases**
- Polimorfismo y sustitución de métodos
- Interfaces
- Clase Object

ORACLE

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

Comportamientos comunes

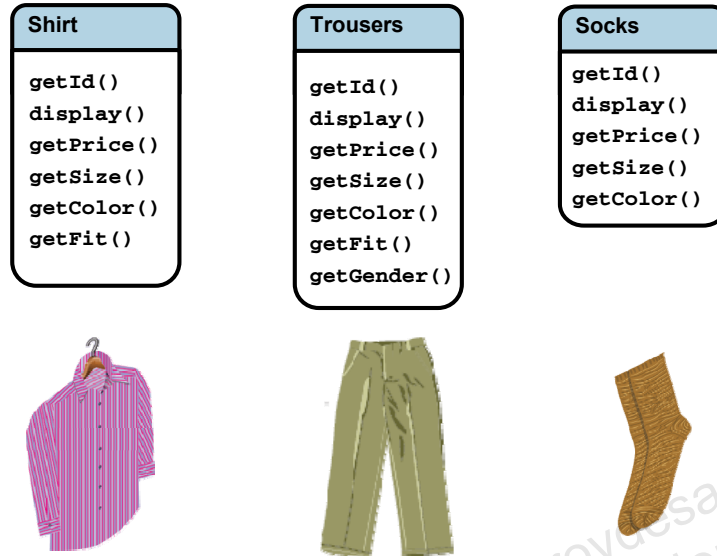
Shirt	Trousers
<code>getId()</code> <code>getPrice()</code> <code>getSize()</code> <code>getColor()</code> <code>getFit()</code>	<code>getId()</code> <code>getPrice()</code> <code>getSize()</code> <code>getColor()</code> <code>getFit()</code> <code>getGender()</code>
<code>setId()</code> <code>setPrice()</code> <code>setSize()</code> <code>setColor()</code> <code>setFit()</code>	<code>setId()</code> <code>setPrice()</code> <code>setSize()</code> <code>setColor()</code> <code>setFit()</code> <code>setGender()</code>
<code>display()</code>	<code>display()</code>

ORACLE

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

La tabla de la diapositiva muestra un juego de comportamientos de la clase `Shirt` y de una nueva clase: `Trousers`. Las clases se muestran completamente encapsuladas para que se pueda acceder a todos los valores de campo sólo mediante los métodos setter y getter. Observe que ambas clases utilizan varios métodos comunes; esto puede provocar una duplicación del código, lo que dificulta el mantenimiento y la expansión posterior y aumenta la posibilidad de que se produzcan errores.

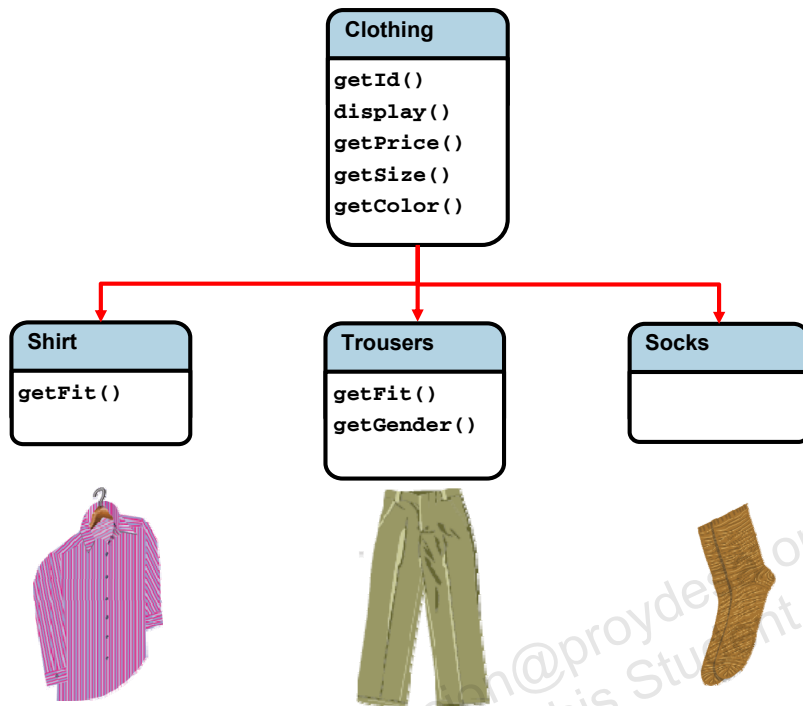
Duplicación de código

**ORACLE**

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

Si Duke's Choice decide agregar un tercer elemento, Socks, además de Trousers y Shirt, puede producirse una duplicación de código aún mayor. El diagrama de la diapositiva sólo muestra los métodos getter para acceder a las propiedades de los nuevos objetos.

Herencia



ORACLE

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

Puede eliminar la duplicación de código de las clases mediante la implantación de la herencia. La herencia permite a los programadores colocar los miembros comunes (campos y métodos) en una clase (la superclase) y permitir que otras clases (las subclases) hereden los miembros comunes de esta nueva clase.

Un objeto instanciado de una subclase se comporta como si los campos y métodos de dicha subclase estuvieran en el objeto. Por ejemplo, se puede instanciar la clase **Trousers** y llamar al método `display()` incluso si la clase **Trousers** no contiene un método `display()`; se heredará de la clase **Clothing**.

Sustitución de métodos de superclase

Puede que los métodos que existen en la superclase:

- No estén implantados en la subclase.
 - El método declarado en la superclase se utiliza en tiempo de ejecución.
- Estén implantados en la subclase.
 - El método declarado en la subclase se utiliza en tiempo de ejecución.

ORACLE

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

Las subclases pueden implantar métodos que ya tienen implantaciones en la superclase. En este caso, las implantaciones de método en la subclase sustituyen la implantación de método de la superclase. Por ejemplo, aunque el campo `colorCode` (y sus métodos de acceso) esté en la superclase, las opciones de color pueden ser diferentes en cada subclase. Por lo tanto, puede que sea necesario sustituir los métodos `get` y `set` para este campo en las subclases individuales.

Superclase Clothing: 1

```
public class Clothing {  
    // Fields  
    private int itemID = 0; // Default ID for all clothing items  
    private String description = "-description required-"; // default  
    private char colorCode = 'U'; //'U' is Unset  
    private double price = 0.0; // Default price for all items  
  
    // Constructor  
    public Clothing(int itemID, String description, char colorCode,  
        double price) {  
        this.itemID = itemID;  
        this.description = description;  
        this.colorCode = colorCode;  
        this.price = price; }  
}
```

ORACLE

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

El código de la diapositiva muestra los campos y el constructor de la superclase Clothing.

Superclase Clothing: 2

```
public void display() {
    System.out.println("Item ID: " + getItemID());
    System.out.println("Item description: " + description);
    System.out.println("Item price: " + getPrice());
    System.out.println("Color code: " + getColorCode());
} // end of display method
public String getDescription(){
    return description;
}
public double getPrice() {
    return price;
}
public int getItemID() {
    return itemID;
}
```

ORACLE

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

El código de la diapositiva muestra los métodos de la superclase Clothing.

Superclase Clothing: 3

```
public char getColorCode() {  
    return colorCode;  
}  
public void setItemID(int itemID) {  
    this.itemID = itemID;  
}  
public void setDescription(String description) {  
    this.description = description;  
}  
public void setColorCode(char colorCode) {  
    this.colorCode = colorCode;  
}  
public void setPrice(double price) {  
    this.price = price;  
}
```

ORACLE

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

El código de la diapositiva muestra los métodos restantes de la superclase Clothing.

Declaración de una subclase

Sintaxis:

```
[class_modifier] class class_identifier extends superclass_identifier
```

ORACLE

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

Declaración de una subclase (palabras clave **extends**, **super** y **this**)

```
public class Shirt extends Clothing {

    private char fit = 'U'; //'U' is Unset, other codes 'S', 'M', or 'L'

    public Shirt(int itemID, String description, char colorCode,
                double price, char fit) {
        super(itemID, description, colorCode, price);
        this.fit = fit;
    }

    public char getFit() {
        return fit;
    }

    public void setFit(char fit) {
        this.fit = fit;
    }
}
```

Se asegura de que Shirt hereda los miembros de Clothing.

super es una referencia a los métodos y atributos de la superclase.

this es una referencia a este objeto.

ORACLE

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

La diapositiva muestra el código de la subclase `Shirt`. El código declara los atributos y métodos que son únicos en esta clase. Los atributos y métodos que son comunes a la clase `Clothing` son heredados y es necesario declararlos.

También incluye dos palabras clave útiles y muestra una forma común de implantar constructores en una subclase.

`super` hace referencia a la superclase. Incluso si se ha sustituido un método de la superclase en la subclase, con la palabra clave `super` podrá llamar al método de la superclase. En el ejemplo de la diapositiva, se utiliza para llamar al constructor de la superclase. Al utilizar esta técnica, se puede llamar al constructor de la superclase para definir todos los atributos comunes del objeto que se está construyendo. A continuación, como en el ejemplo, se pueden definir atributos adicionales en las siguientes sentencias.

El único atributo adicional que tiene `Shirt` es el atributo `fit` y está definido después de llamar al constructor de la superclase. Observe el uso de la palabra clave `this`. En comparación con la palabra clave `super`, `this` es una referencia al objeto de esta clase. No es necesario utilizarla en el ejemplo de la diapositiva, pero es habitual que se utilice en los constructores para facilitar la lectura del código.

Declaración de una subclase: 2

```
//This method overrides display in the Clothing superclass
public void display() {
    System.out.println("Shirt ID: " + getItemID());
    System.out.println("Shirt description: " + description);
    System.out.println("Shirt price: " + getPrice());
    System.out.println("Color code: " + getColorCode());
    System.out.println("Fit: " + getFit());
} // end of display method

// This method overrides the methods in the superclass
public void setColorCode(char colorCode) {

    ... include code here to check that correct codes used ...

    this.colorCode = colorCode;
}
}
```

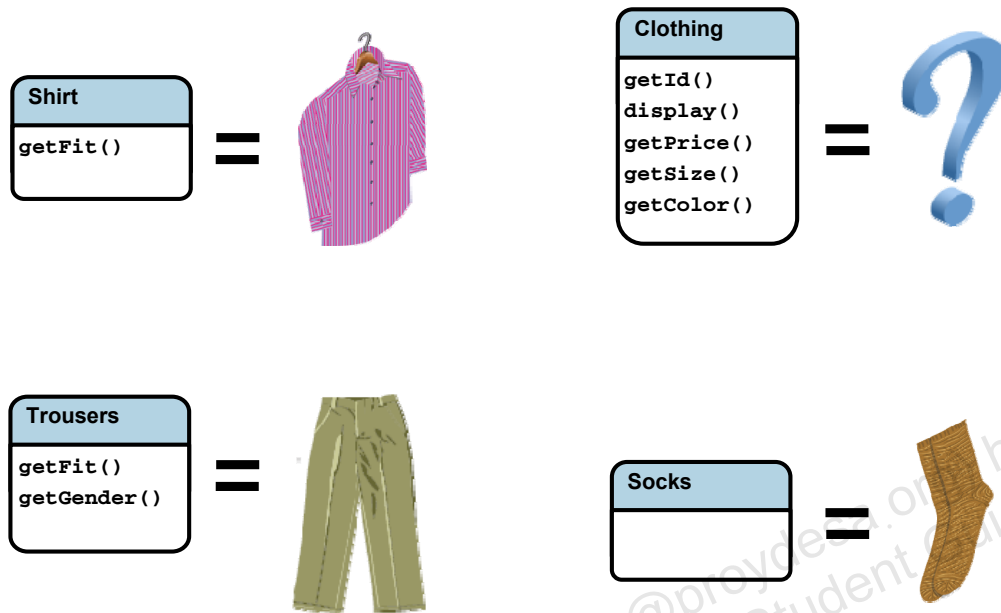
ORACLE

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

Observe que el método `display()` sustituye el método `display()` de la superclase y es más específico de la clase `Shirt`.

Del mismo modo, el método `setColorCode()` sustituye el método `setColorCode()` de la superclase para comprobar si se está utilizando un valor válido para esta clase. (El código no aparece aquí, pero recuerde que se trata de una de las ventajas de la encapsulación de los campos, como se describe en la lección “Uso de encapsulación y constructores”).

Clases abstractas



ORACLE

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

A veces, una superclase tiene sentido como un objeto, pero hay ocasiones en que no es así. Duke's Choice fabrica camisas, calcetines y pantalones, pero no tiene un elemento individual denominado "ropa". Del mismo modo, en la aplicación, puede que la superclase `Clothing` declare algunos métodos que se necesiten en cada subclase (y, por lo tanto, pueden estar en la superclase), pero en realidad no se pueden implantar en la superclase.

Superclase abstracta Clothing: 1

```
public abstract class Clothing {
    // Fields
    private int itemID = 0; // default ID for all clothing items
    private String description = "-description required-"; // default
    private char colorCode = 'U'; //
    private double price = 0.0; // all items

    // Constructor
    public Clothing(int itemID, String description, char colorCode,
        double price, int quantityInStock) {
        this.itemID = itemID;
        this.description = description;
        this.colorCode = colorCode;
        this.price = price;
    }
}
```

La palabra clave **abstract** asegura que la clase no se puede instanciar.

ORACLE

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

Abstracción hace referencia a la creación de clases que son generales y que pueden contener métodos sin una implantación concreta o sin un código de cuerpo de método.

Un ejemplo de una clase abstracta es la clase Clothing tal y como está codificada en esta diapositiva y en las siguientes. Clothing es un concepto abstracto que puede hacer referencia a cualquier cosa. (Normalmente, una persona no va a una tienda y dice “Quiero comprar un artículo de ropa”).

Sin embargo, todos los elementos de ropa tienen algunas características similares en el contexto de un sistema de introducción de pedidos, como un ID o un método para mostrar información sobre el artículo. Las clases que son genéricas y no se pueden definir por completo (como una clase Item) se pueden denominar clases *abstractas*. Las clases que amplían una clase abstracta debe implantar los métodos vacíos de la clase abstracta con un código específico de la subclase. Debe emplear más tiempo en el análisis y el diseño para asegurarse de que la solución tiene una abstracción suficiente para asegurar la flexibilidad.

Superclase abstracta Clothing: 2

```
public abstract char getColorCode() ;

public abstract void setColorCode(char colorCode);

... other methods not listed ...

}
```

La palabra clave `abstract` asegura que estos valores se deben sustituir en la subclase.

ORACLE

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

Los métodos `get` y `set` del campo `colorCode` son abstractos para asegurarse de que se implantan correctamente en cada subclase.

Observe que la subclase `Shirt` mostrada anteriormente se compilará correctamente como una subclase de esta clase abstracta porque ya tiene implantaciones de estos dos métodos. Pero si las implantaciones de `getColorCode()` y `setColorCode()` se eliminan de la subclase `Shirt`, la compilación fallará porque los métodos abstractos de la superclase se deben implantar en la subclase.

Relaciones de superclases y subclases

Es muy importante tener en cuenta el uso más adecuado de la herencia:

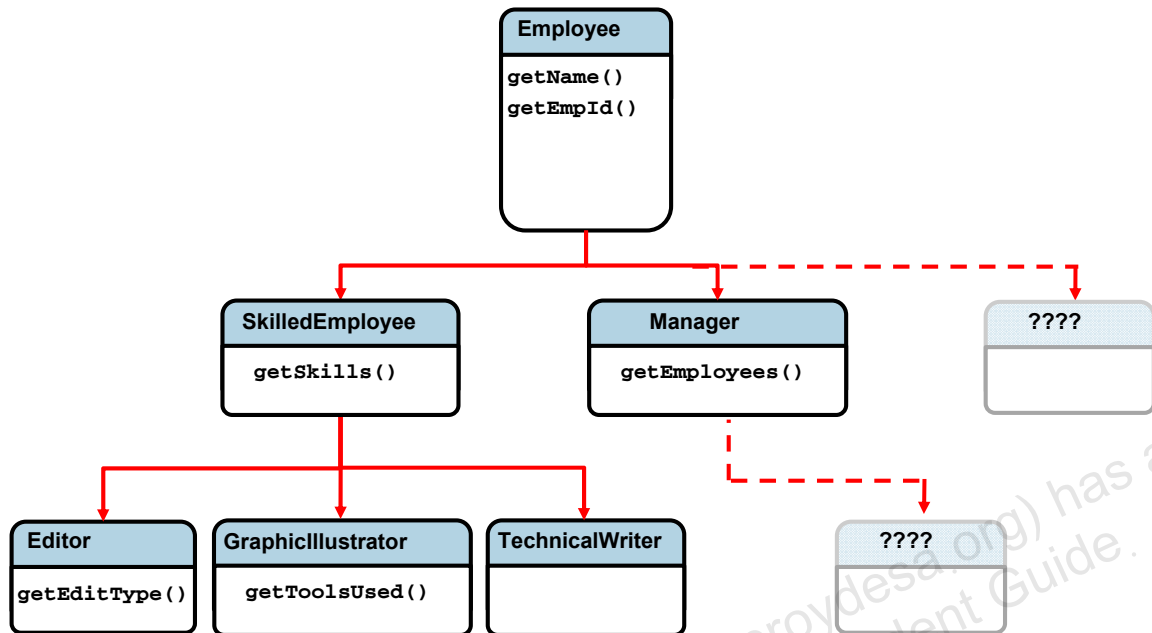
- Utilice la herencia sólo cuando sea completamente válida o inevitable.
- Compruebe si es adecuada con la frase “*es un/una*”:
 - La frase “una camisa es un tipo de ropa” expresa un enlace de herencia válido.
 - La frase “un sombrero es un calcetín” expresa un enlace de herencia no válido.

ORACLE

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

En los ejemplos de este curso, las camisas, los pantalones, los sombreros y los calcetines son todos tipos de ropa. Por lo tanto, Clothing es un nombre adecuado para la superclase de estas subclases (tipos) de ropa.

Otro ejemplo de herencia



ORACLE

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

Esta diapositiva muestra un ejemplo de otro juego de superclases y subclases. En este caso, existen más de dos niveles. La superclase base es **Employee** y actualmente tiene dos subclases. Una de las ventajas de la herencia es que facilita el hecho de que en el futuro se cree una nueva clase que amplíe **Employee** y que dicha clase herede todas las funcionalidades que **Employee** tiene.

Una de las subclases de **Employee** es **SkilledEmployee** y el diagrama muestra que tiene tres subclases propias: **Editor**, **GraphicIllustrator** y **TechnicalWriter**.

Ninguna de estas clases es abstracta. Un empleado y algunos procesos de una aplicación que utilizan estas clases pueden funcionar con la clase **Employee**.

Temas

- Visión general de la herencia
- Trabajar con superclases y subclases
- **Polimorfismo y sustitución de métodos**
- Interfaces
- Clase Object

ORACLE

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

Tipos de referencia de superclase

Hasta ahora hemos analizado la clase utilizada como el tipo de referencia para el objeto creado:

- Para utilizar la clase Shirt como el tipo de referencia para el objeto Shirt:
`Shirt myShirt = new Shirt();`
- Sin embargo, también puede utilizar la superclase como la referencia:

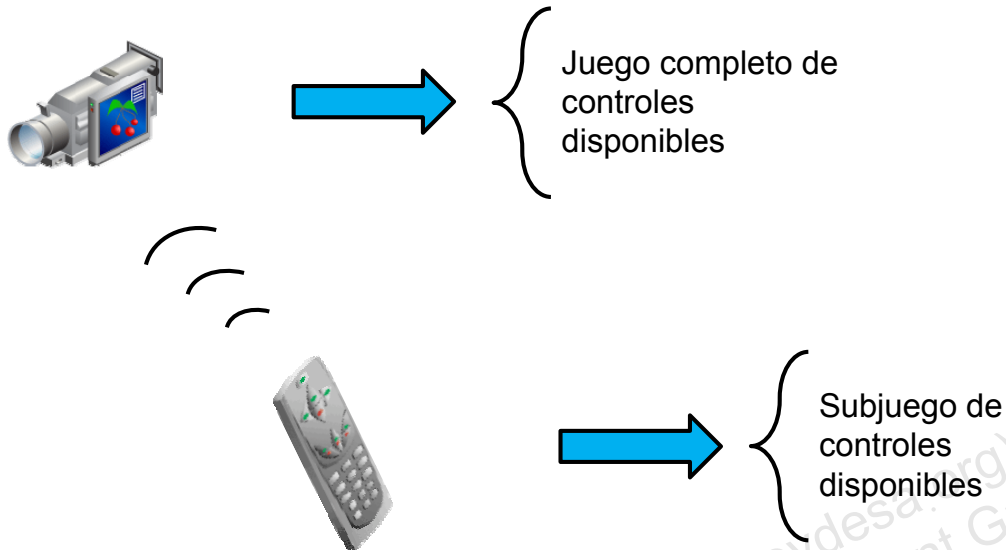
```
Clothing clothingItem1 = new Shirt();  
Clothing clothingItem2 = new Trousers();
```

ORACLE

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

Una importante función de Java es la capacidad de utilizar no sólo la clase en sí, sino también cualquier superclase de la clase como su tipo de referencia. En el ejemplo de la diapositiva, observe que puede hacer referencia tanto al objeto Shirt como al objeto Trousers con la referencia Clothing. Esto significa que una referencia al objeto Shirt o Trousers se puede transferir a un método que necesite una referencia Clothing. O bien, una matriz de Clothing puede contener referencias a los objetos Shirt, Trousers o Socks.

Acceso a funcionalidades de objeto



ORACLE

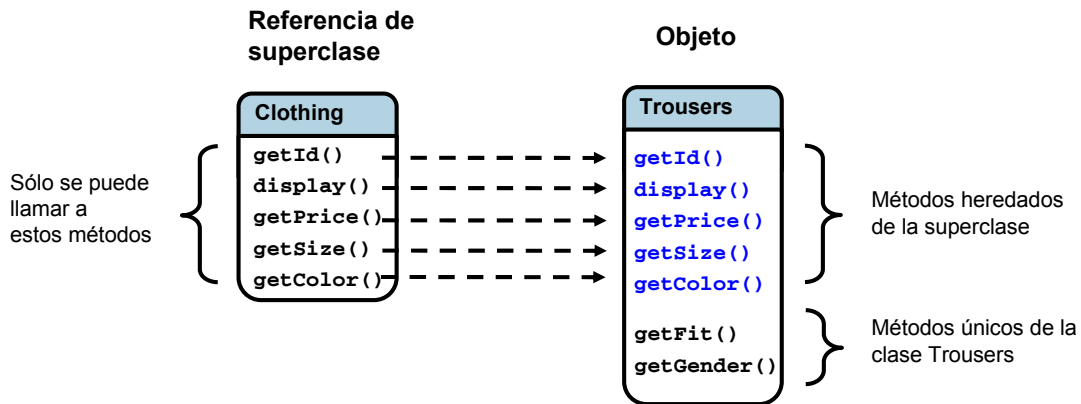
Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

El acceso a los métodos de una clase con una referencia de superclase es como acceder a los controles de un dispositivo electrónico mediante el control remoto en lugar de utilizar los controles propios del dispositivo. Un dispositivo como una cámara de vídeo suele tener un juego completo de controles para grabar, reproducir, editar y acceder a cada una de las funciones disponibles de la cámara. Esto es parecido a utilizar la clase del objeto como el tipo de referencia.

Para algunas combinaciones de cámara de vídeo y control remoto, éste puede proporcionar exactamente los mismos controles, lo que también puede ocurrir al utilizar una superclase como la referencia para un objeto (la superclase le proporciona acceso a todos los métodos del objeto; la clase del objeto no agrega ningún método nuevo). Sin embargo, es posible que el control remoto no tenga el juego completo de controles disponibles en la propia cámara. Una vez más, esto es muy común cuando se utiliza la superclase como referencia.

La superclase sólo tiene acceso a los métodos del objeto que están declarados en la superclase incluso si el objeto tiene otros métodos.

Acceso a métodos de clase desde la superclase

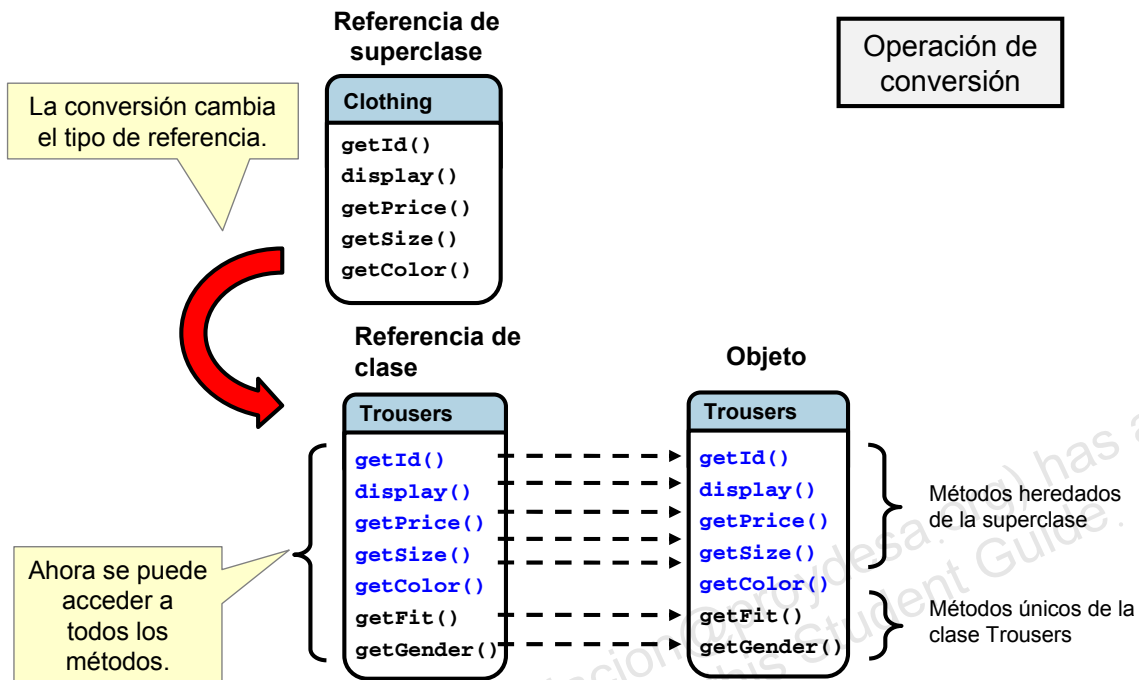


ORACLE

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

El uso de un tipo de referencia `Clothing` no permite acceder al método `getFit()` o `getGender()` del objeto `Trouser`. Normalmente, esto no es un problema porque probablemente esté transfiriendo las referencias `Clothing` a los métodos que no necesitan acceso a estos métodos. Por ejemplo, un método `purchase()` puede recibir un argumento `Clothing` porque sólo necesita acceder al método `getPrice()`.

Conversión del tipo de referencia



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

Ya que una superclase no puede acceder a todos los métodos del objeto al que está haciendo referencia, ¿cómo se puede acceder a estos métodos? Puede hacerlo mediante la sustitución de la referencia de superclase por:

- Una referencia que sea del mismo tipo que el objeto.
- Una interfaz que declare los métodos y se implante mediante la clase de objeto.

(Las interfaces se tratarán en el siguiente tema de esta lección).

Conversión

```
Clothing cl = new Trousers(123, "Dress Trousers", 'B', 17.00, 4, 'S');  
cl.display();  
  
//char fitCode = cl.getFit(); // This won't compile  
  
char fitCode = ((Trousers)cl).getFit(); // This will compile
```

Los paréntesis de `cl` aseguran que la conversión se aplica a esta referencia.

Sintaxis de la conversión: el tipo que se va a convertir se coloca entre paréntesis, antes de la referencia que se va a convertir.

ORACLE

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

El código de este ejemplo muestra la referencia `Clothing` que se va a convertir a una referencia `Trousers` para acceder al método `getFit()`, al cual no se puede acceder a través de la referencia `Clothing`. Observe que los paréntesis internos de `Trousers` forman parte de la sintaxis de conversión, mientras que los paréntesis externos de `((Trousers)cl)` se introducen para aplicar la conversión al tipo `Clothing`.

Operador instanceof

Posible error de conversión:

```
public static void displayDetails(Clothing cl) {

    cl.display();
    char fitCode = ((Trousers) cl).getFitCode();
    System.out.println("Fit: " + fitCode);
}
```

El operador `instanceof` se utiliza para asegurarse de que no existe ningún error de conversión:

```
public static void displayDetails(Clothing cl) {
    cl.display();
    if (cl instanceof Trousers) {
        char fitCode = ((Trousers) cl).getFitCode();
        System.out.println("Fit: " + fitCode);
    }
    else { // Take some other action }
```

El operador `instanceof` devuelve `true` si el objeto al que hace referencia `cl` es un objeto `Trousers`.

ORACLE

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

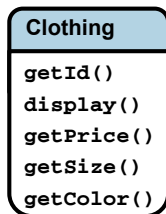
El primer ejemplo de código de la diapositiva muestra un método que está diseñado para recibir un argumento de tipo `Clothing` y, a continuación, convertirlo a `Trousers` para llamar a un método que sólo existe en un objeto `Trousers`. Sin embargo, no es posible saber a qué tipo de objeto apunta la referencia `cl`. Si fuera, por ejemplo, `Shirt`, al intentar convertirlo, se produciría un problema. (Se devolvería una excepción `CastClassException`. La devolución de excepciones se trata en la lección “Manejo de errores”).

Puede evitar este problema potencial con el código mostrado en el segundo ejemplo de código de la diapositiva. Aquí, se utiliza el operador `instanceof` para asegurarse de que `cl` hace referencia a un objeto de tipo `Trousers` antes de intentar realizar la conversión.

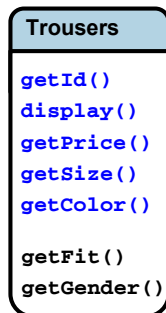
Si cree que el código requiere conversión, tenga en cuenta que suelen existir formas de diseñar código para que la conversión no sea necesaria y, de hecho, es preferible. Sin embargo, si necesita realizar la conversión, debe utilizar `instanceof` para asegurarse de que la conversión no devuelve una excepción `CastClassException`.

Llamadas a métodos polimórficos

Referencia de superclase



Referencia de clase



Independientemente del tipo de referencia utilizado, el método ejecutado está en el objeto instanciado.

Objeto



Métodos heredados de la superclase

Métodos heredados de la superclase y sustituidos

Métodos únicos de la clase Trousers

ORACLE

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

El comportamiento polimórfico que muestra una sentencia puede llamar a uno de los métodos de Clothing. Se trata de una llamada al método polimórfico porque la llamada no necesita o necesita saber el tipo de objeto (a veces denominado el tipo *en tiempo de ejecución*), pero se llamará al método correcto, es decir, al método del objeto real. En el ejemplo de la diapositiva, el objeto es Trousers, pero podría ser cualquier subclase de Clothing.

Prueba

¿Cómo se puede cambiar el tipo de referencia de un objeto?

- a. Mediante la llamada a `getReference()`
- b. Mediante conversión
- c. Mediante la declaración de una nueva referencia y la asignación del objeto

ORACLE

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

Respuesta: b

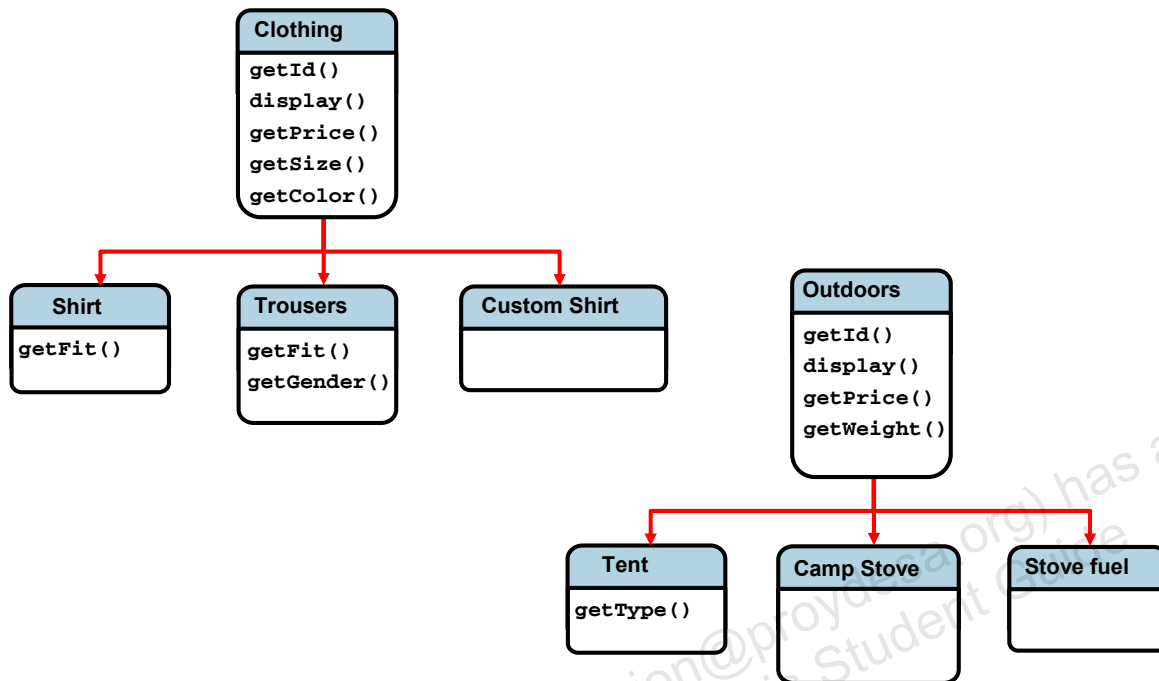
Temas

- Visión general de la herencia
- Trabajar con superclases y subclases
- Polimorfismo y sustitución de métodos
- **Interfaces**
- Clase Object

ORACLE

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

Varías jerarquías



ORACLE

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

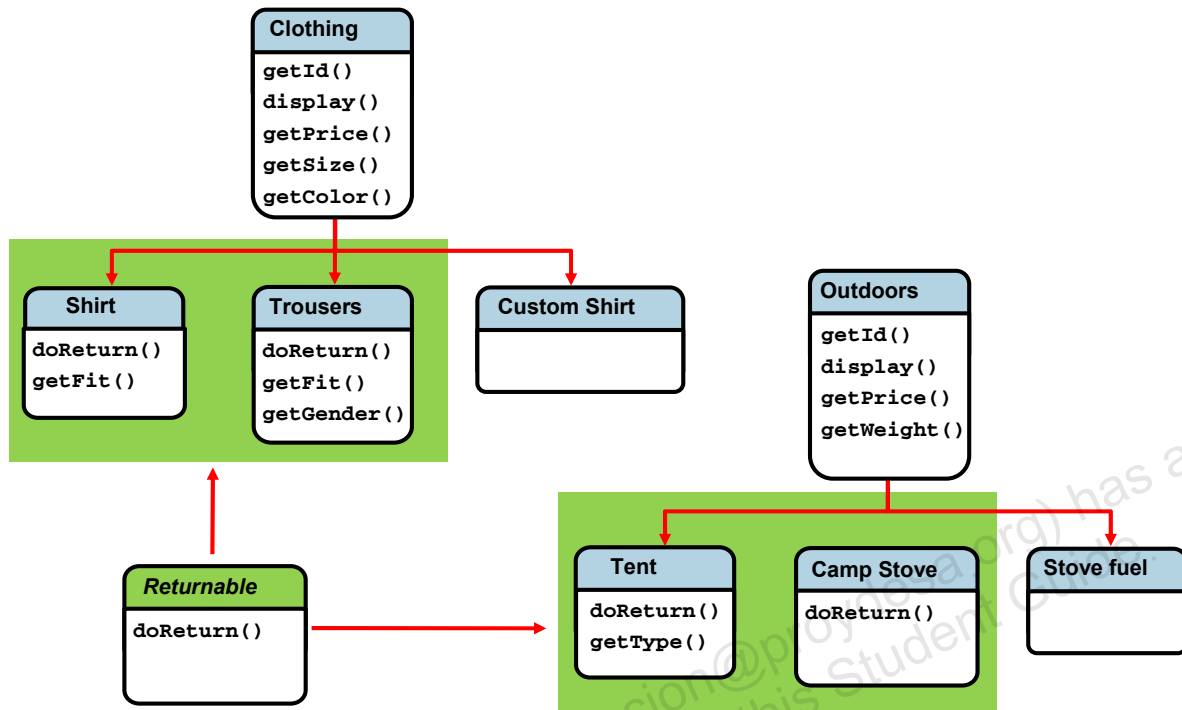
Un juego de clases más complejo tiene elementos en dos jerarquías diferentes. Si Duke's Choice empieza a vender material para exteriores, tiene una superclase que es completamente diferente y que se denomina *Outdoors*, con su propio juego de subclasses (por ejemplo, `getWeight()` como un método de *Outdoors*).

En este supuesto, pueden existir algunas clases de cada jerarquía que tengan algo en común. Por ejemplo, el elemento *Custom Shirt* de *Clothing* no se puede devolver (porque está fabricado a mano para una persona en concreto), al igual que el elemento *Stove Fuel* de la jerarquía *Outdoors*. El resto de elementos sí se pueden devolver.

¿Cómo se puede modelar? A continuación, se indican algunos aspectos a tener en cuenta:

- Una nueva superclase no funcionará porque una clase sólo puede ampliar una superclase y todos los elementos amplían actualmente *Outdoors* o *Clothing*.
- Se podría utilizar un nuevo campo denominado *Returnable*, agregado a cada clase, para determinar si un elemento se puede devolver. Esto es posible, pero entonces no existiría ningún tipo de referencia único para transferir a un método que inicia o procesa una devolución.
- Puede utilizar un tipo especial denominado *interfaz* que cualquier clase puede implantar. Este tipo se puede utilizar para transferir una referencia de cualquier clase que lo implante.

Interfaces



ORACLE

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

El diagrama de la diapositiva muestra todos los elementos que se pueden devolver mediante la implantación de la interfaz `Returnable` con un método único, `doReturn()`. Los métodos se pueden declarar en una interfaz, pero no se pueden implantar en ella. Por lo tanto, cada clase que implanta `Returnable` debe implantar `doReturn()` para sí misma. Todos los elementos que se pueden devolver se pueden transferir al método `processReturns()` de una clase `Returns` y, a continuación, pueden llamar a su método `doReturn()`.

Implantación de la interfaz Returnable

Interfaz Returnable

```
public interface Returnable {
    public String doReturn();
}
```

Al igual que un método abstracto, sólo tiene el stub de método.

Asegura que Shirt debe implantar todos los métodos de Returnable.

Clase Shirt

```
public class Shirt extends Clothing implements Returnable {
    public Shirt(int itemID, String description, char colorCode,
        double price, char fit) {
        super(itemID, description, colorCode, price);
        this.fit = fit;
    }
    public String doReturn() {
        // See notes below
        return "Suit returns must be within 3 days";
    }
    ...< other methods not shown > ... } // end of class
```

Método declarado en la interfaz Returnable

ORACLE

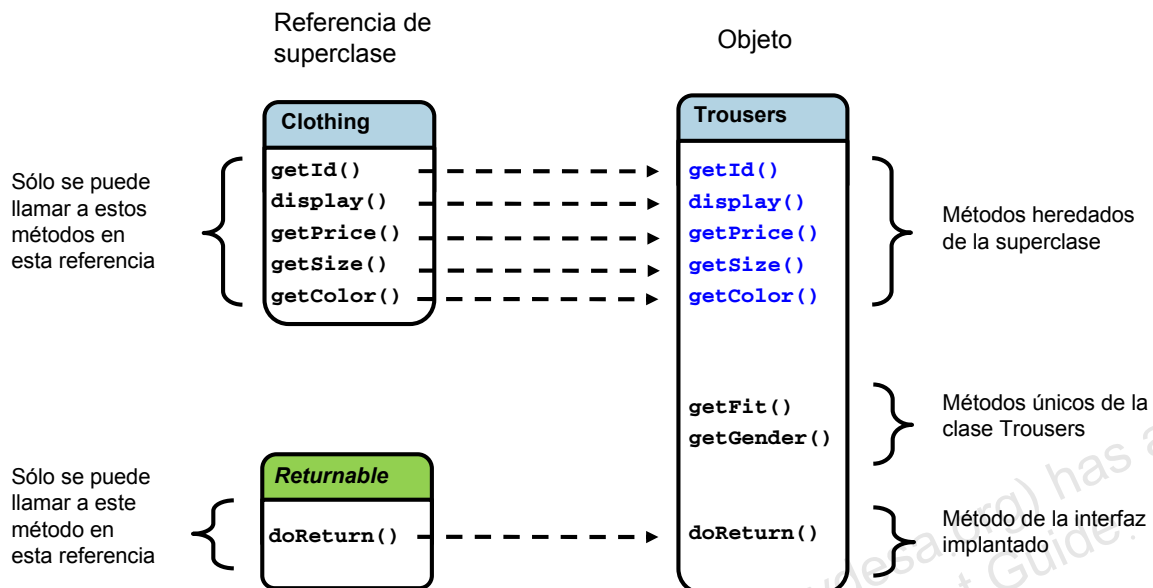
Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

El código de este ejemplo muestra la interfaz `Returnable` y la clase `Shirt`. Sólo se muestran el constructor y el método `doReturn()`.

En esta implantación, `Returnable` proporciona un marcador para indicar que el elemento se puede devolver y se asegura que el desarrollador de `Shirt` debe implantar el método `doReturn()`.

El método `doReturn()` devuelve una cadena que describe las condiciones para devolver el elemento.

Acceso a los métodos de objeto desde la interfaz



ORACLE

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

Como se muestra en una diapositiva anterior, la referencia utilizada para acceder a un objeto determina los métodos a los que se puede llamar en ella. En caso de la referencia de interfaz mostrada en la diapositiva, sólo se puede llamar al método `getReturn()`. Si un método recibe una referencia `Returnable` y necesita acceder a los métodos de `Clothing` o `Trousers`, la referencia se puede convertir al tipo de referencia adecuado.

ArrayList

ArrayList se amplía desde AbstractList que, a su vez, se amplía desde AbstractCollection.

The screenshot shows the Java API documentation for the `ArrayList<E>` class. The page is titled "Class ArrayList<E>" and shows its inheritance hierarchy: `java.lang.Object` (parent), `java.util.AbstractCollection<E>` (parent), `java.util.AbstractList<E>` (parent), and `java.util.ArrayList<E>` (current class). The "All Implemented Interfaces" section lists `Serializable`, `Cloneable`, `Iterable<E>`, `Collection<E>`, `List<E>`, and `RandomAccess`. The "Direct Known Subclasses" section lists `AttributeList`, `RoleList`, and `RoleUnresolvedList`. The class declaration is shown as `public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess, Cloneable, Serializable`. A detailed description states: "Resizable-array implementation of the List interface. Implements all optional list operations, and permits all elements, including null. In addition to implementing the List interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This class is roughly equivalent to Vector, except that it is unsynchronized.)"

ArrayList implanta una serie de interfaces.

La interfaz List es la que se utiliza principalmente al trabajar con ArrayList.

ORACLE

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

Algunos de los ejemplos más adecuados de la herencia y la utilidad de los tipos abstractos y de interfaz se pueden consultar en la API de Java. Por ejemplo, la clase `ArrayList` amplía la clase `AbstractList` que, a su vez, amplía `AbstractCollection`. `AbstractCollection` implanta la interfaz `List`, lo cual significa que `ArrayList` también implanta la interfaz `List`.

Para utilizar `ArrayList` como `List`, utilice la interfaz `List` como el tipo de referencia.

Interfaz List

Overview Package **Class** Use Tree Deprecated Index Help

Prev Class Next Class Frames No Frames

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.util

Interface List<E>

Type Parameters:
E - the type of elements in this list

All Superinterfaces:
Collection<E>, Iterable<E>

All Known Implementing Classes:
AbstractList, AbstractSequentialList, ArrayList, AttributeList, CopyOnWriteArrayList, LinkedList, RoleList, RoleUnresolvedList, Stack, Vector

```
public interface List<E>
    extends Collection<E>
```

An ordered collection (also known as a *sequence*). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list.

Existen varias
clases que
implantan la
interfaz List

ORACLE

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

La interfaz List la implantan varias clases. Esto significa que a cualquier método que necesite una interfaz List en realidad se le puede transferir una referencia List a cualquier objeto de estos tipos (pero no las clases abstractas, ya que no se pueden instanciar).

Temas

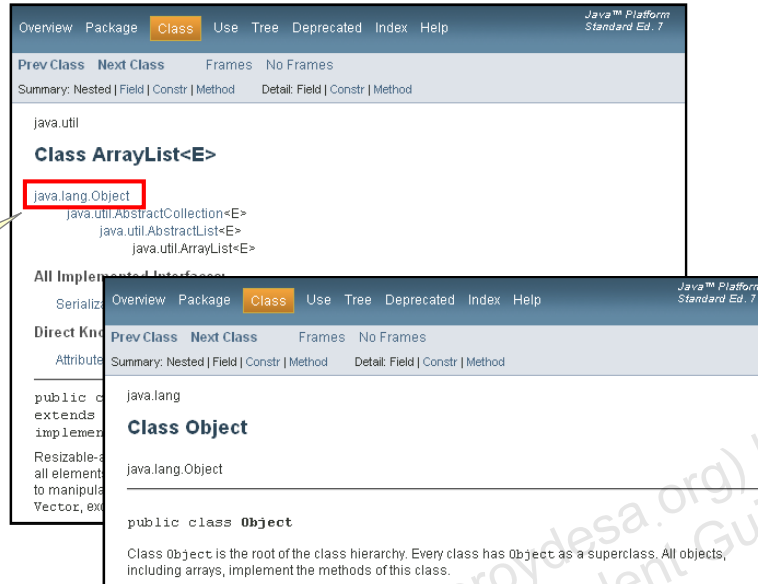
- Visión general de la herencia
- Trabajar con superclases y subclases
- Polimorfismo y sustitución de métodos
- Interfaces
- **Clase Object**

ORACLE®

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

Clase Object

La clase Object es la clase base.



ORACLE

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

Todas las clases tienen en la parte superior de su jerarquía la clase `Object`. Es muy importante saber cómo funciona Java, ya que todas las clases que no amplíen de forma explícita otra clase, ampliarán automáticamente `Object`.

Por lo tanto, todas las clases tienen `Object` como raíz de su jerarquía. Esto significa que todas las clases tienen acceso a los métodos de `Object`. Como raíz de la jerarquía de objetos, `Object` no tiene ningún método, sólo los básicos que todo objeto debe tener.

Un método interesante es el método `toString()`. El método `Object toString()` proporciona información básica sobre el objeto; las clases generales sustituirán el método `toString()` para proporcionar una salida más útil. `System.out.println()` utiliza el método `toString()` en un objeto que se le ha transferido para crear una representación de cadena.

Llamada al método toString()

Se utiliza toString() de Object.

StringBuilder sustituye toString() de Object.

First hereda toString() de Object.

Second sustituye toString() de Object.

```

1 public class Main {
2     public static void main(String[] args) {
3
4         // Output an Object to the console
5         System.out.println(new Object());
6
7         // Output this StringBuilder object to the console
8         System.out.println(new StringBuilder("Some text for StringBuilder"));
9
10        //Output a class that does not override the toString() method
11        System.out.println(new First());
12
13        //Output a class that "does" override the toString() method
14        System.out.println(new Second());
15    }
16 }

```

Output - TestCode (run)

```

java.lang.Object@3e25a5
Some text for StringBuilder
First@19821f
This class named Second has overridden the toString() method of Object
BUILD SUCCESSFUL (total time: 1 second)

```

Salida de las llamadas al método toString() de cada objeto.

ORACLE

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

Todos los objetos tienen un método toString() porque éste existe en la clase Object. Sin embargo, el método toString() puede devolver resultados diferentes, según si se ha sustituido o no. En el ejemplo de la diapositiva, se llama a toString() (mediante el método println() de System.out) en cuatro objetos:

- **Un objeto Object:** se llama al método toString() de la clase base. Devuelve el nombre de la clase (java.lang.Object), un símbolo @ y un valor hash del objeto (un número único asociado al objeto).
- **Un objeto StringBuilder:** se llama al método toString() en el objeto StringBuilder. StringBuilder sustituye el método toString() que hereda de Object para devolver un objeto String del juego de caracteres que representa.
- **Un objeto de tipo First, una clase de prueba:** First es una clase sin código, por lo que el método toString() llamado es el que se hereda de la clase Object.
- **Un objeto de tipo Second, una clase de prueba:** Second es una clase con un método denominado toString(), por lo que este método sustituido será el método al que se llame.

Sería conveniente volver a implantar el método getDescription() utilizado por las clases de Clothing en lugar de utilizar un método toString() sustituido.

Prueba

¿A qué métodos de un objeto se puede acceder mediante una interfaz que implante?

- a. A todos los métodos implantados en la clase del objeto
- b. A todos los métodos implantados en la superclase del objeto
- c. A los métodos declarados en la interfaz

ORACLE

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

Respuesta: c

Resumen

En esta lección debe haber aprendido lo siguiente:

- La creación de jerarquías de clase con subclases y superclases ayuda a crear código extensible y que se puede mantener mediante:
 - La generalización y abstracción de código que, de lo contrario, se duplicaría.
 - El uso del polimorfismo.
- La creación de interfaces:
 - Permite enlazar clases en diferentes jerarquías de objetos mediante su comportamiento común.
 - Permite el uso de un tipo de referencia de interfaz en el código para que la clase de implantación se pueda cambiar con mayor facilidad.



ORACLE

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

La herencia permite a los programadores colocar los miembros comunes (variables y métodos) en una clase y permitir que otras clases *hereden estos miembros comunes* de esta nueva clase.

La clase que contiene los miembros comunes a varias clases se denomina *superclase* o *clase principal*. Las clases que heredan de la superclase o la amplían se denominan *subclases* o *clases secundarias*.

La herencia también permite hacer referencia a los campos y métodos de objeto mediante una referencia que es el tipo del objeto, el tipo de cualquiera de sus superclases o una interfaz que implanta.

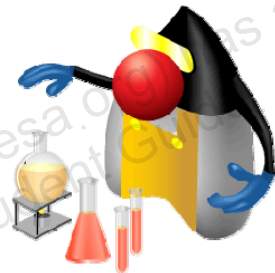
Por último, la herencia permite el polimorfismo.

Visión general de la práctica 12-1: Creación y uso de superclases y subclasses

En esta práctica, diseñará y creará una jerarquía de clases que formará la base de un sistema de seguimiento de empleados del departamento de marketing de la compañía Duke's Choice.

En esta práctica, podrá:

- Crear un modelo de diseño sencillo para la jerarquía de clases
- Crear las clases reales y probarlas

**ORACLE**

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

Visión general de la práctica 12-2: Uso de una interfaz Java

En esta práctica, creará una interfaz. En esta práctica, podrá:

- Crear una interfaz denominada Printable e implantarla en la jerarquía de clases creada en la práctica 11-1
- Examinar y ejecutar otra pequeña aplicación que utilice la misma interfaz Printable

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.

13

Manejo de errores

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 los diferentes tipos de errores que se pueden producir y cómo se manejan en Java
- Describir las excepciones que se utilizan en Java
- Determinar las excepciones que se devuelven para cualquiera de las clases base
- Escribir código para manejar una excepción devuelta por el método de una clase base



ORACLE

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

Temas

- Manejo de errores: visión general
- Propagación de excepciones
- Captura y devolución de excepciones
- Varias excepciones y errores

ORACLE

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

Informe de excepciones

Error de código:

```
int[] intArray = new int[5];  
intArray[5] = 27;
```

Salida en la consola:

```
Exception in thread "main"  
    java.lang.ArrayIndexOutOfBoundsException: 5  
        at TestErrors.main(TestErrors.java:17)
```

ORACLE

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

Puede que aparezca el error que se muestra en la diapositiva cuando trabaje en algunas de las actividades de la práctica anterior. El código muestra un error común producido al acceder a una matriz. Recuerde que las matrices están basadas en cero (se accede al primer elemento mediante un índice cero), por lo tanto, en una matriz como la que aparece en la diapositiva que tiene cinco elementos, el último elemento en realidad es `intArray[4]`.

`intArray[5]` intenta acceder a un elemento que no existe y Java responde a este error de programación con la impresión del texto que aparece en la consola.

Informe de excepciones

Código de llamada en main():

```
TestArray myTestArray = new TestArray(5);
myTestArray.addElement(5, 23);
```

Clase TestArray:

```
public class TestArray {
    int[] intArray;
    public TestArray (int size) {
        intArray = new int[size];
    }
    public void addElement(int index, int value) {
        intArray[index] = value;
    }
}
```



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

A continuación aparece un ejemplo muy similar. Esta vez, el código que crea la matriz e intenta asignar un valor a otro elemento no existente se ha movido a una clase diferente. Observe que el mensaje de error de la consola es casi idéntico al del ejemplo anterior pero, en esta ocasión, aparecen los métodos `main()` de `TestException` y `addElement()` de `TestArray`.

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
    at TestArray.addElement(TestArray.java:19)
    at TestException.main(TestException.java:20)
```

Java Result: 1

En esta lección, conocerá el motivo por el que el mensaje se imprime en la consola. También aprenderá a capturar o interrumpir el mensaje para que no se imprima en la consola, además de conocer otros tipos de errores de Java.

Devolución de excepciones

Ejecución normal del programa:

1. El método de llamada llama al método de trabajo.
2. El método de trabajo no funciona.
3. El método de trabajo termina el trabajo y, a continuación, la ejecución se devuelve al método de llamada.

Cuando se produce una excepción, la siguiente secuencia cambia:

- La excepción se devuelve y:
 - Se transfiere un objeto Exception especial a un bloque catch similar a un método especial en el método actual o bien
 - La ejecución se devuelve al método de llamada

ORACLE

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

Tipos de excepciones

Existen tres tipos principales de objetos Throwable:

- **Error**
 - Normalmente, un error externo irrecuperable
 - No comprobada
- **RuntimeException**
 - Normalmente, un error de programación
 - No comprobada
- **Exception**
 - Error recuperable
 - Comprobada (capturada o devuelta)

ORACLE

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

Como se menciona en la diapositiva anterior, al devolver una excepción, ésta es un objeto que se puede transferir a un bloque catch. Existen tres tipos principales de objetos que se pueden devolver de esta forma y todos están derivados de la clase Throwable.

OutOfMemoryError

Error de programación:

```
ArrayList theList = new ArrayList();
while(true) {
    String theString = "A test String";
    theList.add(theString);
    if (theList.size()% 1000000 == 0) {
        System.out.println("List now has " +
            theList.size()/100000 + " million elements!");
    }
}
```

Salida en la consola:

```
List now has 240 million elements!
List now has 250 million elements!
Exception in thread "main" java.lang.OutOfMemoryError: Java
heap space
```

ORACLE

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

OutOfMemoryError es un error. Los objetos Throwable de tipo Error se suelen utilizar para condiciones excepcionales que son externas a la aplicación, la cual no las puede anticipar ni recuperarse de ellas de forma habitual.

El ejemplo mostrado tiene un bucle infinito que agrega continuamente un elemento a una ArrayList, lo que garantiza que JVM se quedará sin memoria. El error se devuelve en la pila de llamadas y, puesto que no se captura en ninguna ubicación, aparece en la consola de la siguiente forma:

```
List now has 240 million elements!
List now has 250 million elements!
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at java.util.Arrays.copyOf(Arrays.java:2760)
    at java.util.Arrays.copyOf(Arrays.java:2734)
    at java.util.ArrayList.ensureCapacity(ArrayList.java:167)
    at java.util.ArrayList.add(ArrayList.java:351)
    at TestErrors.main(TestErrors.java:22)
```

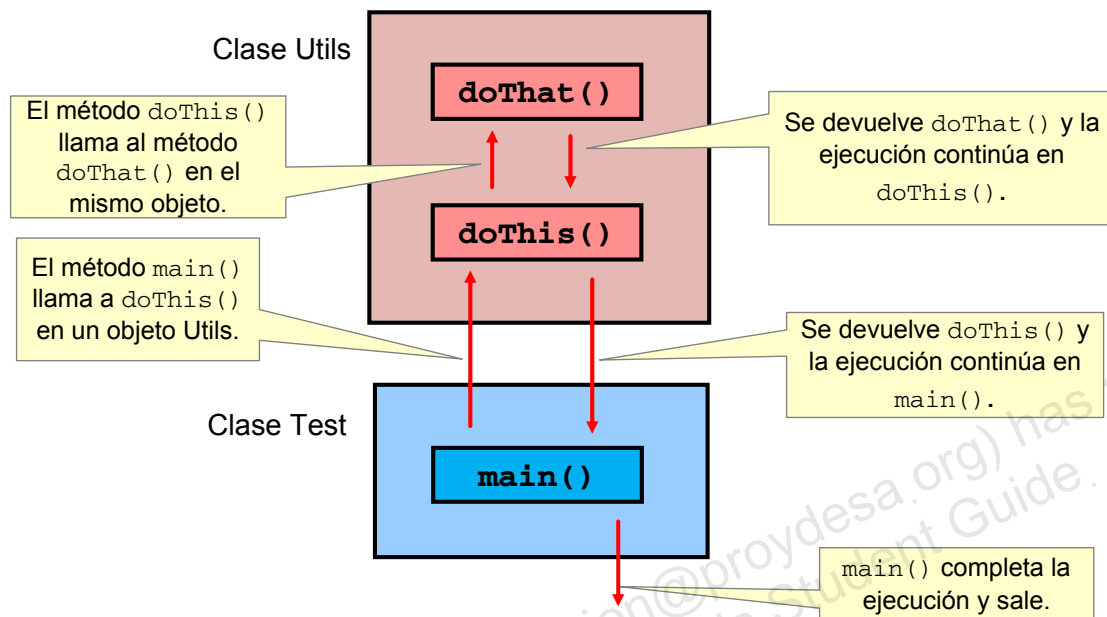
Temas

- Manejo de errores: visión general
- **Propagación de excepciones**
- Captura y devolución de excepciones
- Varias excepciones y errores

ORACLE

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

Pila de métodos



ORACLE

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

Para comprender las excepciones, debe pensar sobre cómo los métodos llaman a otros métodos y la forma en la que estas acciones se pueden anidar en profundidad. El modo de funcionamiento normal es que un método de llamada llame a un método de trabajo, que se convierte en un método de llamada y llama a otro método de trabajo y así sucesivamente. Esta secuencia de métodos se denomina *pila de llamadas*.

El ejemplo que aparece en la diapositiva ilustra tres métodos en esta relación. El método principal de la clase `Test` (un método estático) instancia un objeto de tipo `Utils` y llama al método `doThis()` en ese objeto. El método `doThis()` llama a un método privado `doThat()` en el mismo objeto. Al llegar al final de su código o a una sentencia de retorno, cada método devuelve la ejecución al método que lo llamó.

Observe que, en cuanto a la forma en la que se llama a los métodos y éstos se devuelven y la forma en la que se devuelven las excepciones, el hecho de que aquí sólo exista un método de clase y dos métodos de instancia en el mismo objeto resulta irrelevante.

Pila de llamadas: Ejemplo

Clase Test:

```
public static void main (String args[]) {  
    Utils theUtils = new Utils();  
    theUtils.doThis();  
}
```

Clase Utils:

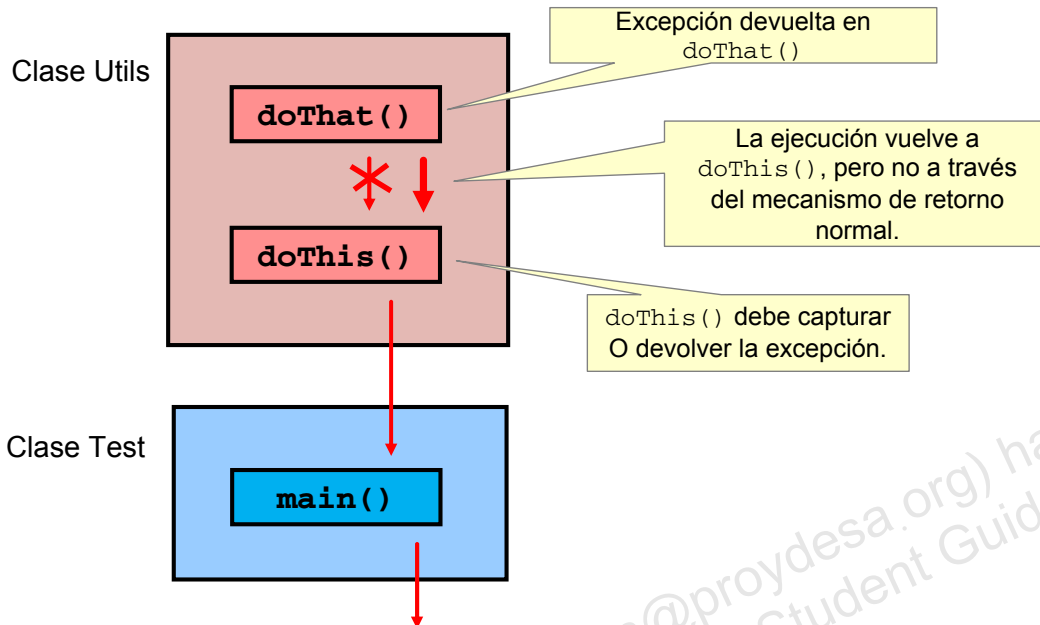
```
public void doThis() {  
    ...< code to do something >...  
    doThat();  
    return;  
  
    public void doThat() throws Exception{  
        ...< code to do something >...  
        if (some_problem) throw new Exception();  
        return;  
    }  
}
```

ORACLE

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

El código mostrado en esta diapositiva es el posible código para el ejemplo ilustrado en la diapositiva anterior.

Devolución de objetos Throwable



ORACLE

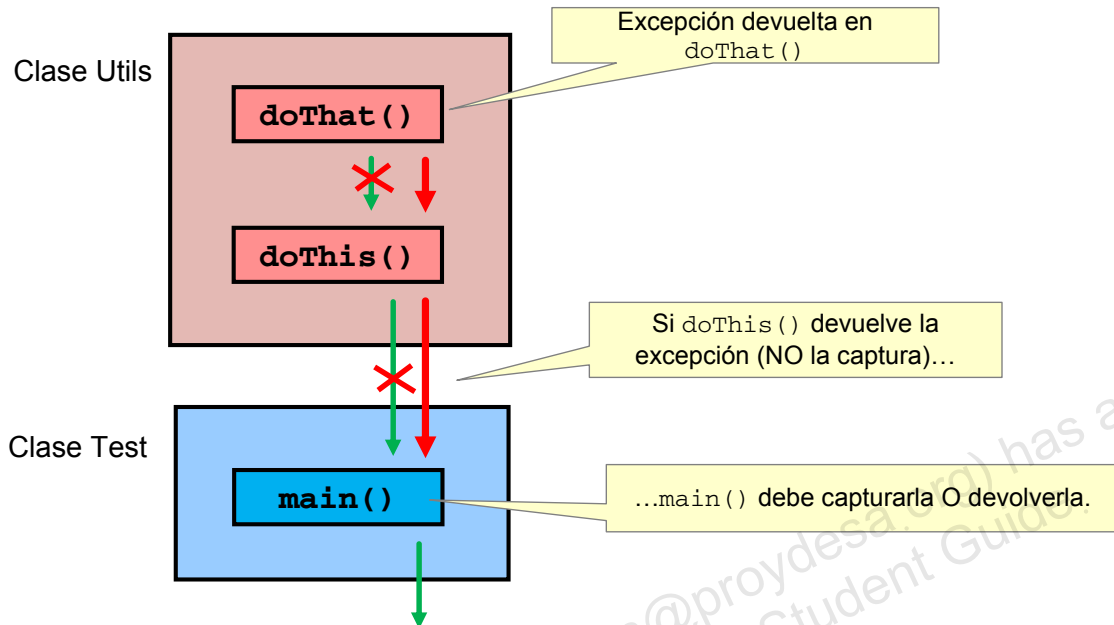
Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Cuando un método termina de ejecutarse, el flujo normal (cuando se termina el método o en una sentencia de retorno) vuelve al método de llamada y continúa la ejecución en la siguiente línea del método de llamada.

Cuando se devuelve una excepción, el flujo de programa vuelve al método de llamada, pero no al punto inmediatamente posterior a la llamada al método. En su lugar, si existe un bloque try/catch, éste se devuelve al bloque catch que está asociado al bloque try que contiene la llamada al método. Si no hay ningún bloque try/catch en el método de llamada, la excepción se devuelve al método de llamada.

En el caso de una excepción comprobada, esto ocurre porque se fuerza al programador a devolver explícitamente la excepción si no decide capturarla. En el caso de una excepción que es una clase `RuntimeException` o un error, la devolución de la excepción se produce automáticamente cuando no hay bloques try/catch.

Devolución de objetos Throwable



ORACLE

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

El diagrama de la diapositiva ilustra una excepción originalmente devuelta en `doThat()` que se está devolviendo a `doThis()`. El error no se captura aquí, por lo que se devuelve a su método de llamada, que es el método principal.

Trabajar con excepciones en NetBeans

```

10 public class Utils {
11
12     public void doThis() {
13
14         System.out.println("Arrived in doThis()");
15         doThat();
16         System.out.println("Back in doThis()");
17     }
18
19
20     public void doThat() {
21         System.out.println("In doThat()");
22     }
23 }
24

```

No se devuelve ninguna excepción; no es necesario realizar ninguna acción.

NetBeans utiliza una ayuda de burbuja que le proporcionará dos opciones.

La devolución de una excepción en el método necesita pasos adicionales.

```

12 public void doThis() {
13
14     System.out.println("Arrived in doThis()");
15     doThat();
16     System.out.println("Back in doThis()");
17 }
18
19
20 public void doThat() {
21     System.out.println("In doThat()");
22     throw new Exception();
23 }
24
25

```

unreported exception java.lang.Exception;
must be caught or declared to be thrown
(Alt-Enter shows hints)

ORACLE

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

Aquí puede consultar el código de la clase Utils mostrado en NetBeans. En la primera captura de pantalla, no se devuelve ninguna excepción, por lo que NetBeans no muestra ningún error de compilación ni de sintaxis. En la segunda captura de pantalla, `doThat()` devuelve una excepción y NetBeans la marca como un elemento que el programador debe manejar. Como puede observar en la ayuda de burbuja, ésta proporciona dos opciones que el programador debe elegir para manejar las excepciones comprobadas.

En estos ejemplos se utiliza la superclase `Exception` para que resulte más sencillo. Sin embargo, como podrá ver más adelante, por lo general, no debe devolver una excepción. Cuando sea posible, si captura una excepción, debe intentar capturar una excepción específica.

Captura de una excepción

```

12 public void doThis() {
13
14     System.out.println("Arrived in doThis()");
15     doThat();
16     // unreported exception java.lang.Exception;
17     // must be caught or declared to be thrown
18 }
19
20 public void doThat() throws Exception {
21     System.out.println("In doThat()");
22     throw new Exception();
23 }
24
25

```

Ahora se debe manejar la excepción en doThis().

doThat() ahora devuelve una excepción.

El bloque try/catch captura una excepción y la maneja.

```

12 public void doThis() {
13
14     System.out.println("Arrived in doThis()");
15     try {
16         doThat();
17     }
18     catch (Exception e) {
19         System.out.println(e);
20     }
21     System.out.println("Back in doThis()");
22 }
23
24 public void doThat() throws Exception {
25     System.out.println("In doThat()");
26     throw new Exception();
27 }
28

```

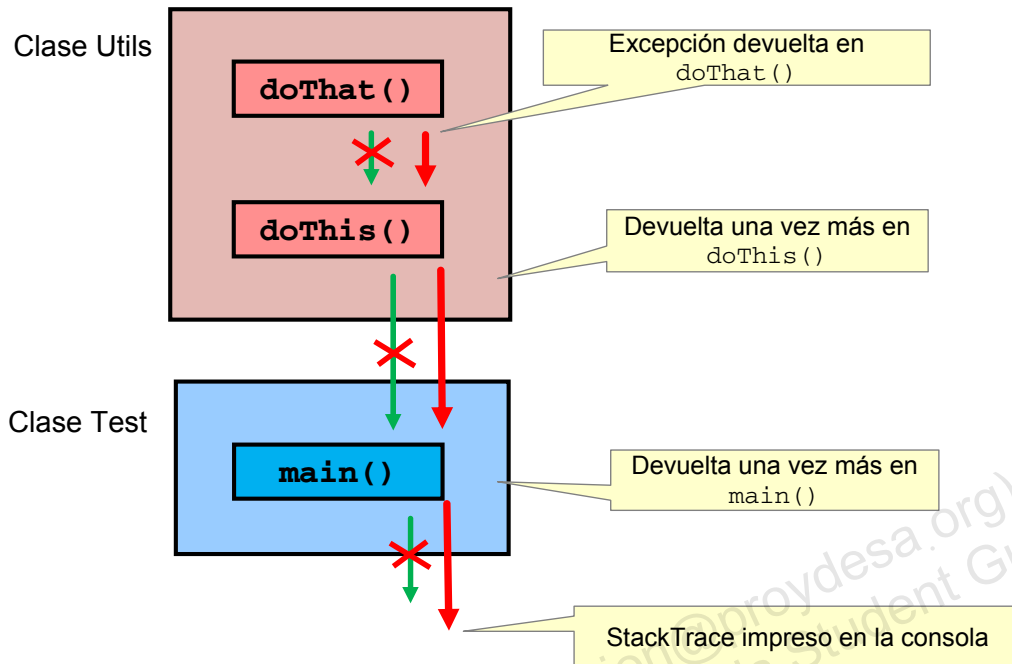
ORACLE

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

Aquí puede observar que la excepción devuelta en doThat() se ha manejado de la siguiente forma:

- Mediante la adición de throws Exception a la firma de método doThat(), asegurándose de que ésta se devuelve al emisor de llamada, doThat()
- Mediante la adición de un bloque try/catch a doThis() de forma que:
 - El bloque try contenga la llamada a doThat()
 - El bloque catch se configure con el parámetro Exception

Excepción no resuelta



ORACLE

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

¿Qué ocurre si ninguno de los métodos de la pila de llamadas tiene bloques try/catch? Esta situación se ilustra en el diagrama mostrado en la diapositiva. Debido a que no hay bloques try/catch, la excepción se devuelve al inicio de la pila de llamadas. Pero, ¿qué significa devolver una excepción desde el método `main()`? Esto provoca que el programa se cierre y que la excepción, además de un rastreo de pila de la misma, se imprima en la consola.

Excepción impresa en la consola

Ejemplo de `main()` devolviendo una excepción.

The screenshot shows a Java IDE with two panels. The top panel displays the source code of a class named `Test`. The `main` method is highlighted with a red box and a callout stating: "main() ahora está definido para devolver una excepción." The code is as follows:

```

10 public class Test {
11
12     public static void main (String args[]) throws Exception {
13
14         System.out.println("Started in main()");
15         Utils myUtils = new Utils();
16         myUtils.doThis();
17         System.out.println("Back in main()");
18     }
19
20 }

```

The bottom panel shows the 'Output - TestCode (run)' window. It displays the execution output and a stack trace for an exception. A red box highlights the stack trace, with a callout stating: "Puesto que main() devuelve la excepción, ahora imprime una pila de llamadas en la consola." The output is as follows:

```

run:
Started in main()
Arrived in doThis()
In doThat()
Exception in thread "main" java.lang.Exception
    at Utils.doThat(Utils.java:27)
    at Utils.doThis(Utils.java:16)
    at Test.main(Test.java:16)
Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)

```

ORACLE

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

En el ejemplo, puede ver lo que ocurre cuando la excepción se devuelve a la pila de llamadas hasta el método `main()` y se devuelve también la excepción.

¿Ha observado que es similar al primer ejemplo de `ArrayIndexOutOfBoundsException`? En ambos casos, la excepción aparece como un rastreo de pila en la consola.

Sin embargo, existe una diferencia en `ArrayIndexOutOfBoundsException`: ninguno de los métodos ha devuelto la excepción. ¿Cómo ha recorrido la pila de llamadas?

Porque `ArrayIndexOutOfBoundsException` es una clase `RuntimeException`. La clase `RuntimeException` es una subclase de la clase `Exception`. Tiene la funcionalidad adicional de que sus excepciones se devuelven automáticamente a la pila de llamadas sin que esté declarada explícitamente en la firma de método.

Resumen de los tipos de excepciones

Throwable es un tipo especial de objeto Java:

- Es el único tipo de objeto que se utiliza como argumento en una cláusula catch.
- Es el único tipo de objeto que se puede “devolver” al método de llamada.
- Tiene dos subclases:
 - Error
 - Si se crea, se devuelve automáticamente al método de llamada.
 - Exception
 - Se debe devolver explícitamente al método de llamada.
 - - Se obtiene mediante el uso de un bloque try/catch.
 - Tiene una subclase RuntimeException que se devuelve automáticamente al método de llamada.

ORACLE

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

Las excepciones que no son RuntimeExceptions se deben manejar explícitamente. Los ejemplos que aparecen más adelante en esta lección le mostrarán cómo trabajar con una excepción IOException.

Prueba

¿Cuál de las siguientes afirmaciones es cierta?

- a. Una excepción RuntimeException se debe capturar.
- b. Una excepción RuntimeException se debe devolver.
- c. Una excepción RuntimeException se debe capturar o devolver.
- d. Una excepción RuntimeException se devuelve automáticamente.

ORACLE

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

Respuesta: d

Prueba

¿Cuáles de los siguientes objetos son excepciones comprobadas?

- a. Todos los objetos de tipo Throwable.
- b. Todos los objetos de tipo Exception.
- c. Todos los objetos de tipo Exception que no son de tipo RuntimeException.
- d. Todos los objetos de tipo Error.
- e. Todos los objetos de tipo RuntimeException.

ORACLE

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

Respuesta: c

Temas

- Manejo de errores: visión general
- Propagación de excepciones
- **Captura y devolución de excepciones**
- Varias excepciones y errores

ORACLE

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

Excepciones en la documentación de la API de Java

Method Summary

Methods

Modifier and Type	Method and Description
boolean	<code>canExecute()</code> Tests whether the application can execute the file denoted by this abstract pathname.
boolean	<code>canRead()</code> Tests whether the application can read the file denoted by this abstract pathname.
boolean	<code>canWrite()</code> Tests whether the application can modify the file denoted by this abstract pathname.
int	<code>compareTo(File pathname)</code> Compares two abstract pathnames lexicographically.
boolean	<code>createNewFile()</code> Atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.

createNewFile

```
public boolean createNewFile()
    throws IOException
```

Atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist. The check for the existence of the file and the creation of the file if it does not exist are a single operation that is atomic with respect to all other filesystem activities that might affect the file.

Note: this method should not be used for file-locking, as the resulting protocol cannot be made to work reliably. The `FileLock` facility should be used instead.

Returns:

true if the named file does not exist and was successfully created; false if the named file already exists

Throws:

- `IOException` - If an I/O error occurred
- `SecurityException` - If a security manager exists and its `SecurityManager.checkWrite(java.lang.String)` method denies write access to the file

Since:

1.2

Haga clic para obtener los detalles de `createNewFile()`.

Observe las excepciones que se pueden devolver.

ORACLE

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

Al trabajar con cualquier API, es necesario determinar las excepciones que devuelven los métodos o constructores del objeto. El ejemplo de la diapositiva es para la clase `File`. `File` tiene un método `createNewFile()` que puede devolver una excepción `IOException` o `SecurityException`. `SecurityException` es una excepción `RuntimeException`, por lo que no está comprobada, pero `IOException` sí es una excepción comprobada.

Llamada a un método que devuelve una excepción

```

31
32
33
34
35
36
37
38
39
40
41
public static void testCheckedException() {
    File testFile = new File("//testFile.txt");

    System.out.println("File exists: " + testFile.exists());
    testFile.delete();
    System.out.println("File exists: " + testFile.exists());
}

```

El constructor no causa ningún problema de compilación.

```

31
32
33
34
35
36
37
38
39
40
41
public static void testCheckedException() {
    File testFile = new File("//testFile.txt");
    testFile.createNewFile();

    System.out.println("File exists: " + testFile.exists());
    testFile.delete();
    System.out.println("File exists: " + testFile.exists());
}

```

`createNewFile()` puede devolver una excepción comprobada, por lo que debe devolverla o capturarla.

ORACLE

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

Las dos capturas de pantalla de la diapositiva muestran un método `testCheckedException()` sencillo. En el primer ejemplo, el objeto `File` se crea con el constructor. Observe que, aunque el constructor pueda devolver una excepción `NullPointerException` (si el argumento del constructor es nulo), no se le forzará a capturar esta excepción.

Sin embargo, en el segundo ejemplo, `createNewFile()` puede devolver una excepción `IOException` y NetBeans le indicará que debe manejarla.

Observe que `File` se introduce aquí sólo para ilustrar una excepción `IOException`. En el siguiente curso (Programación 2), obtendrá más información sobre la clase `File` y un nuevo juego de clases del paquete `nio`, que le proporcionará formas más sofisticadas de trabajar con los archivos.

Trabajar con una excepción comprobada

Captura de IOException:

```
public static void main(String args[]) {
    try {
        testCheckedException();
    }
    catch (IOException e) {
        System.out.println(e);
    }
}

public static void testCheckedException() throws
IOException{
    File testFile = new File("//testFile.txt");
    testFile.createNewFile();
    System.out.println("File exists: " + testFile.exists());
}
```



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

El ejemplo de la diapositiva maneja la posible excepción emitida mediante:

- La devolución de la excepción del método `testCheckedException()`
- La captura de la excepción en el método de llamada

En este ejemplo, el método `catch` captura la excepción porque la ruta de acceso al archivo de texto no tiene el formato correcto. `System.out.println(e)` llama al método `toString()` de la excepción y el resultado es el siguiente:

```
java.io.IOException: The filename, directory name, or volume label
syntax is incorrect
```

Prácticas recomendadas

- Capture la excepción real devuelta, no la excepción ni la superclase Throwable.
- Examine la excepción para conocer el problema exacto y así poder realizar la recuperación correctamente.
- No es necesario que capture todas las excepciones.
 - Un error de programación no se debe manejar. Se debe corregir.
 - Debe preguntarse si esta excepción representa el comportamiento del que desea que se recupere el programa.

ORACLE

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

Prácticas no recomendadas

```

public static void main (String args[]) {
    try {
        createFile("c:/testFile.txt");
    }
    catch (Exception e) {
        System.out.println("Problem creating the file!");
        ...< other actions >...
    }
}

public static void createFile(String fileName) throws
IOException {
    File f = new File(fileName);
    f.createNewFile();

    int[] intArray = new int[5];
    intArray[5] = 27;
}

```

¿Se captura la superclase?

¿Se está procesando el objeto Exception?

ORACLE

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

El código de la diapositiva ilustra dos prácticas de programación deficientes.

1. La cláusula `catch` captura una excepción en lugar de la excepción esperada de la llamada al método `createFile` (`IOException`).
2. La cláusula `catch` no analiza el objeto `Exception` y, en su lugar, simplemente asume que la excepción esperada se ha devuelto del objeto `File`.

La principal desventaja de este estilo de programación descuidada es el hecho de que el código imprime el siguiente mensaje en la consola:

```
There is a problem creating the file!
```

Esto sugiere que el archivo no se ha creado y que, de hecho, se ejecutará cualquier código posterior en el bloque `catch`. Pero, ¿qué es lo que ocurre en realidad en el código?

Prácticas no recomendadas

```

public static void main (String args[]) {
    try {
        createFile("c:/testFile.txt");
    }
    catch (Exception e) {
        System.out.println(e);
        ...< other actions >...
    }
}

public static void createFile(String fileName) throws
IOException {
    File f = new File(fileName);
    System.out.println(fileName + " exists? " + f.exists());
    f.createNewFile();
    System.out.println(fileName + " exists? " + f.exists());
    int[] intArray = new int[5];
    intArray[5] = 27;
}

```

¿Cuál es el tipo de objeto?

Se llama a toString() en este objeto.

ORACLE

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

Si introduce algunas llamadas `System.out.println()` en el método `createFile`, puede resultarle más fácil entender lo que ocurre. Ahora la salida es:

```

C:/testFile.txt exists? false
C:/testFile.txt exists? true
java.lang.ArrayIndexOutOfBoundsException: 5

```

Por tanto, se crea el archivo. Puede observar que la excepción en realidad es una excepción `ArrayIndexOutOfBoundsException` que está devolviendo la línea final de código en `createFile()`.

En este ejemplo, resulta obvio que la asignación de matriz puede devolver una excepción, pero podría no ser tan evidente. En este caso, el método `createNewFile()` de `File` en realidad devuelve otra excepción, `SecurityException`. Ya que se trata de una excepción no comprobada, se devuelve automáticamente.

Si comprueba la excepción específica de la cláusula `catch`, evita el peligro de suponer cuál es el problema.

Temas

- Manejo de errores: visión general
- Propagación de excepciones
- Captura y devolución de excepciones
- **Varias excepciones y errores**

ORACLE

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

Varias excepciones

```

public static void createFile() throws IOException {
    File testF = new File("c:/notWriteableDir");
    File tempF = testF.createTempFile("te", null, testF);
    System.out.println("Temp filename: " +
        tempF.getPath());
    int myInt[] = new int[5];
    myInt[5] = 25;
}

```

El directorio debe permitir la escritura (IOException).

El argumento debe tener tres o más caracteres (IllegalArgumentException).

El índice de matriz debe ser válido (ArrayIndexOutOfBoundsException).

ORACLE

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

El ejemplo de la diapositiva muestra un método que podría devolver tres excepciones diferentes. Utiliza el método `File createTempFile()`, que crea un archivo temporal. (Asegura que cada llamada crea un archivo nuevo y diferente y que también se puede configurar para que los archivos temporales creados se supriman al salir).

Las tres excepciones diferentes son las siguientes:

IOException

`c:\notWriteableDir` es un directorio, pero no permite la escritura. Esto provoca que `createTempFile()` devuelva una `IOException` (comprobada).

IllegalArgumentException

El primer argumento transferido para `createTempFile` debe tener una longitud de tres caracteres o más. Si no es así, el método devuelve `IllegalArgumentException` (no comprobada).

ArrayIndexOutOfBoundsException

Como en los ejemplos anteriores, al intentar acceder a un índice no existente de una matriz, se devuelve una excepción `ArrayIndexOutOfBoundsException` (no comprobada).

Captura de IOException

```
public static void main (String args[]) {
    try {
        createFile();
    }
    catch (IOException ioe) {
        System.out.println(ioe);
    }
}

public static void createFile() throws IOException {

    File testF = new File("c:/notWriteableDir");
    File tempF = testFile.createTempFile("te", null, testF);
    System.out.println("Temp filename is " +
        tempFile.getPath());
    int myInt[] = new int[5];
    myInt[5] = 25;
}
```

ORACLE

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

El ejemplo de la diapositiva muestra el manejo mínimo de excepciones (el compilador insiste en el manejo de, al menos, la excepción IOException).

Con el directorio definido como se muestra en `c:/notWriteableDir`, la salida del código es:

```
java.io.IOException: Permission denied
```

Sin embargo, si el archivo se define como `c:/writeableDir` (un directorio que permite la escritura), la salida será:

```
Exception in thread "main" java.lang.IllegalArgumentException: Prefix
string too short
```

```
    at java.io.File.createTempFile(File.java:1782)
    at
MultipleExceptionExample.createFile(MultipleExceptionExample.java:34)
    at MultipleExceptionExample.main(MultipleExceptionExample.java:18)
```

El argumento "te" provoca que se devuelva una excepción `IllegalArgumentException` y, debido a que se trata de una `RuntimeException`, se devuelve en toda la consola.

Captura de IllegalArgumentException

```
public static void main (String args[]) {
    try {
        createFile();
    }
    catch (IOException ioe) {
        System.out.println(ioe);
    } catch (IllegalArgumentException iae) {
        System.out.println(iae);
    }
}

public static void createFile() throws IOException {

    File testF = new File("c:/writeableDir");
    File tempF = testFile.createTempFile("te", null, testF);
    System.out.println("Temp filename is " + tempFile.getPath());
    int myInt[] = new int[5];
    myInt[5] = 25;
}
```

ORACLE

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

El ejemplo de la diapositiva muestra una cláusula `catch` adicional para capturar la excepción `IllegalArgumentException` potencial.

Con el primer argumento del método `createTempFile()` definido en "te" (menos de tres caracteres), la salida del código es:

```
java.lang.IllegalArgumentException: Prefix string too short
```

Sin embargo, si el argumento se define en "temp", la salida será:

```
Temp filename is /Users/kenny/writeableDir/temp938006797831220170.tmp
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
... < some code omitted > ...
```

Ahora se está creando el archivo temporal, pero el método `createFile()` aún está devolviendo otro argumento. Debido a que `ArrayIndexOutOfBoundsException` es una excepción `RuntimeException`, se devolverá automáticamente a la consola.

Captura de las excepciones restantes

```
public static void main (String args[]) {
    try {
        createFile();
    }
    catch (IOException ioe) {
        System.out.println(ioe);
    } catch (IllegalArgumentException iae) {
        System.out.println(iae);
    } catch (Exception e) {
        System.out.println(e);
    }
}

public static void createFile() throws IOException {
    File testF = new File("c:/writeableDir");
    File tempF = testFile.createTempFile("te", null, testF);
    System.out.println("Temp filename is " + tempFile.getPath());
    int myInt[] = new int[5];
    myInt[5] = 25;
}
```



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

El ejemplo de la diapositiva muestra una cláusula `catch` adicional para capturar todas las excepciones restantes.

Para el código de ejemplo, la salida de este código es:

```
Temp filename is /Users/kenny/writeableDir/temp7999507294858924682.tmp
java.lang.ArrayIndexOutOfBoundsException: 5
```

Por último, la cláusula de la excepción `catch` se puede agregar para capturar cualquier excepción adicional.

Resumen

En esta lección, debe haber aprendido lo siguiente:

- Describir los diferentes tipos de errores que se pueden producir y cómo se manejan en Java
- Describir las excepciones que se utilizan en Java
- Determinar las excepciones que se devuelven para cualquiera de las clases base
- Escribir código para manejar una excepción devuelta por el método de una clase base



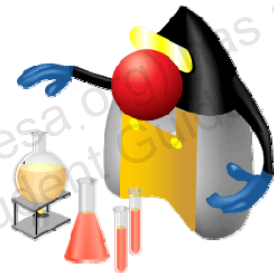
ORACLE

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

Visión general de la práctica 13-1: Uso de un bloque try/catch para manejar una excepción

En esta práctica, manejará una excepción devuelta por el método `parse()` de `SimpleDateFormat`. En esta práctica, podrá:

- Utilizar la documentación de la API de Java para examinar la clase `SimpleDateFormat` y buscar la excepción devuelta por el método `parse()`
- Crear una clase que llame al método `parse()`
- Escribir un bloque try/catch para capturar la excepción devuelta por `parse()`

**ORACLE**

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

Visión general de la práctica 13-2: Captura y devolución de una excepción personalizada

En esta práctica, utilizará una excepción personalizada denominada `InvalidSkillException`. Puede utilizarla con la aplicación de seguimiento de empleados diseñada y creada en las prácticas 12-1 y 12-2.

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.

14

Despliegue y mantenimiento de la aplicación Duke's Choice

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 lo siguiente:

- Desplegar una aplicación simple como un archivo JAR
- Describir las partes de una aplicación Java, lo que incluye la interfaz de usuario y el backend
- Describir cómo se pueden ampliar las clases para implantar nuevas capacidades en la aplicación

**ORACLE**

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

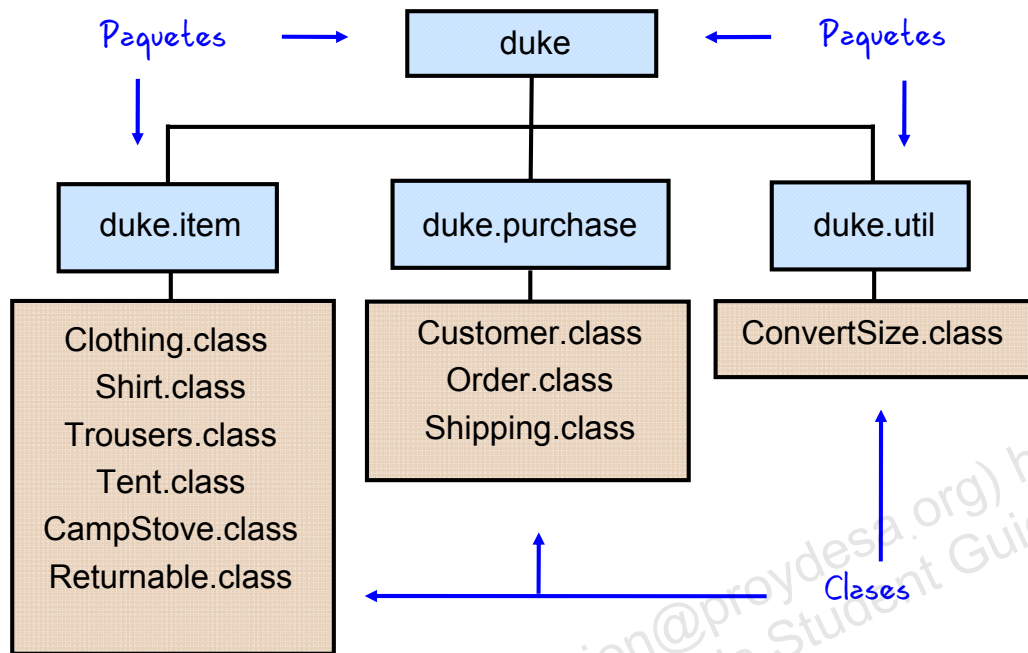
Temas

- Paquetes
 - JAR y despliegue
 - Arquitectura de dos y tres niveles
 - Aplicación Duke's Choice
 - Mejoras y modificaciones de la aplicación

ORACLE

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

Paquetes



ORACLE

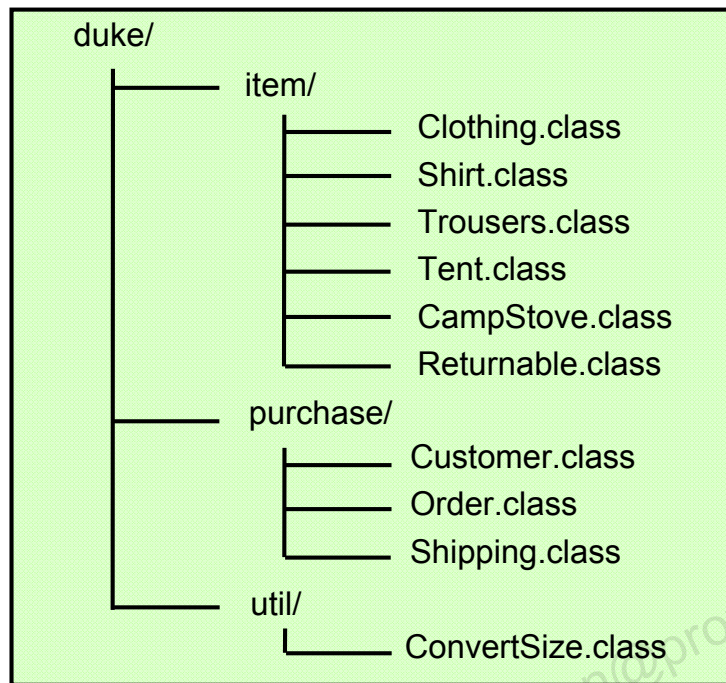
Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Las clases se agrupan en paquetes para facilitar la gestión del sistema.

Existen varias formas de agrupar clases en paquetes significativos. No hay ninguna forma correcta o incorrecta, pero una técnica común es agrupar clases en paquetes por similitud semántica.

Por ejemplo, el software de Duke's Choice puede incluir un juego de clases de artículos (como `Shirt`, `Trousers`, `Tent`, las superclases `Clothing` y `Camping`, etc.), un juego de clases que utiliza estas clases de artículos para organizar compras y un juego de clases de utilidades. Todos estos paquetes se incluyen en el paquete de nivel superior denominado `duke`.

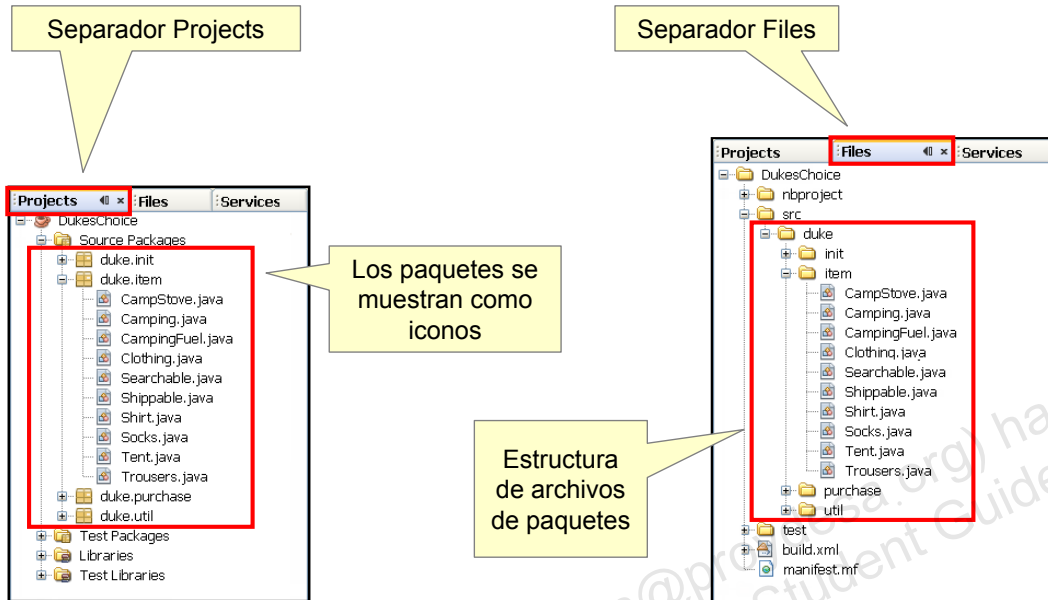
Estructura del directorio de paquetes

**ORACLE**

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

Los paquetes se almacenan en un árbol de directorios que contiene directorios que coinciden con los nombres de paquetes. Por ejemplo, el archivo `Clothing.class` debe existir en el directorio `item`, incluido en el directorio `duke`.

Paquetes en NetBeans



ORACLE

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

El panel izquierdo de NetBeans tiene tres separadores. Dos de ellos, Projects y Files, muestran cómo los paquetes están relacionados con la estructura de archivos.

El separador Projects muestra los paquetes y las bibliotecas de cada proyecto (la captura de pantalla sólo muestra `DukesChoice`). El paquete de origen que se muestra es el que contiene los paquetes y las clases de Duke's Choice y la captura de pantalla muestra los cuatro paquetes `duke.init`, `duke.item`, `duke.purchase` y `duke.util`. Cada uno de estos paquetes se puede ampliar para mostrar los archivos de origen, tal y como se ha hecho en el paquete `duke.item` de la captura de pantalla.

El separador Files muestra la estructura de directorios de cada proyecto. En la captura de pantalla, puede ver cómo los paquetes que aparecen en el separador Projects tienen una estructura de directorios correspondiente. Por ejemplo, el paquete `duke.item` tiene la estructura de archivos correspondiente del directorio `duke` justo bajo el directorio `src` y contiene el directorio `item`, que a su vez contiene todos los archivos de origen del paquete.

Paquetes en el código fuente

Esta clase
está en el
paquete
duke.item.

```
package duke.item;
```

```
public abstract class Clothing implements Searchable, Shippable {  
    private int itemID = 0;  
    private String description = "-description required-";  
    private char colorCode = 'U';  
  
    ... < remaining code omitted > ...  
}
```

El paquete al que pertenece una clase se define en el código fuente.

ORACLE

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

El código de ejemplo de la diapositiva muestra la sentencia package que se está utilizando para definir el paquete en el que está incluida la clase Clothing. Al igual que la clase debe estar incluida en un archivo que tenga el mismo nombre que la clase, el archivo (en este caso, Clothing.java) se debe incluir en una estructura de directorios que tenga el mismo nombre de paquete.

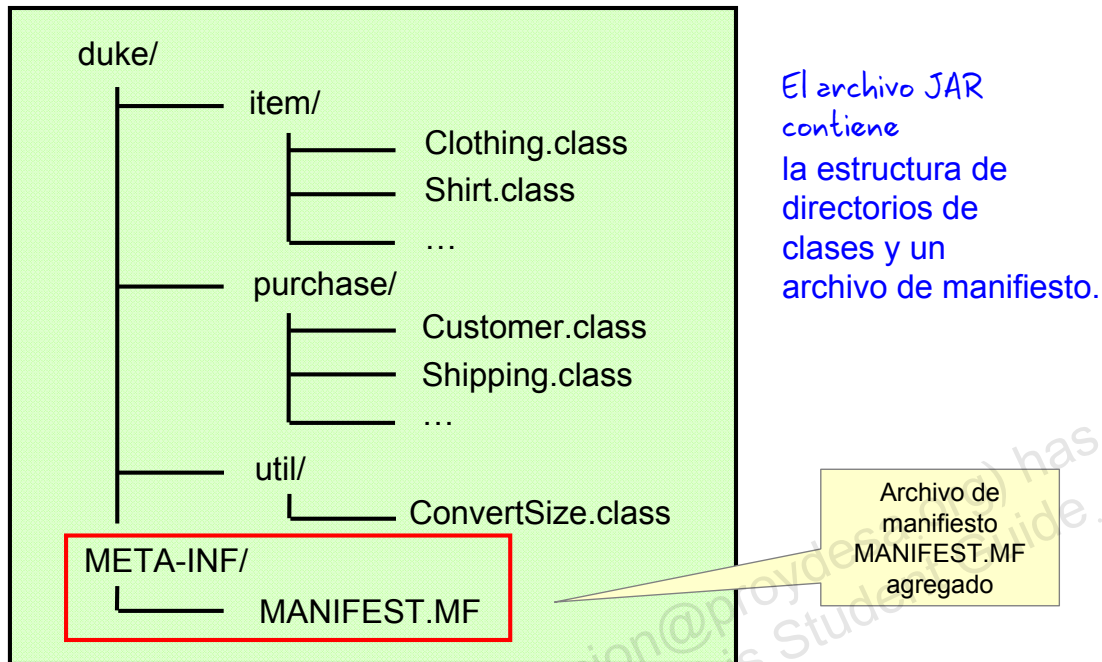
Temas

- Paquetes
- **JAR y despliegue**
- Arquitectura de dos y tres niveles
- Aplicación Duke's Choice
- Mejoras y modificaciones de la aplicación

ORACLE

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

DukesChoice.jar



ORACLE

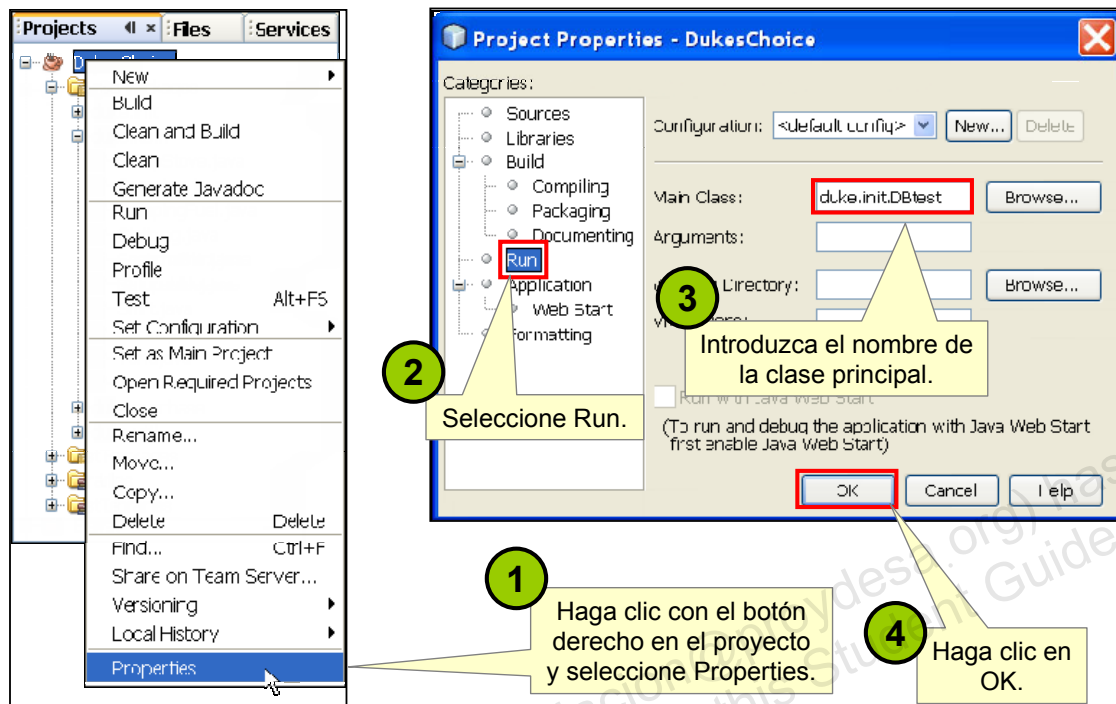
Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Para desplegar una aplicación Java, se suelen incluir los archivos necesarios en un archivo JAR. Esto simplifica en gran medida la ejecución de la aplicación en otra máquina.

Un archivo JAR tiene similitudes con un archivo zip (o un archivo tar en UNIX) y contiene la estructura de directorios completa de las clases compiladas, además de un archivo `MANIFEST.MF` adicional en el directorio `META-INF`. Dicho archivo `MANIFEST.MF` indica al tiempo de ejecución de Java cuál es el archivo que contiene el método `main()`.

Puede crear un archivo JAR mediante el uso de una herramienta de línea de comandos denominada `jar`, aunque la mayoría de los IDE facilitan su creación. En las siguientes diapositivas, puede ver cómo crear un archivo JAR con NetBeans.

Definición de la clase principal de un proyecto

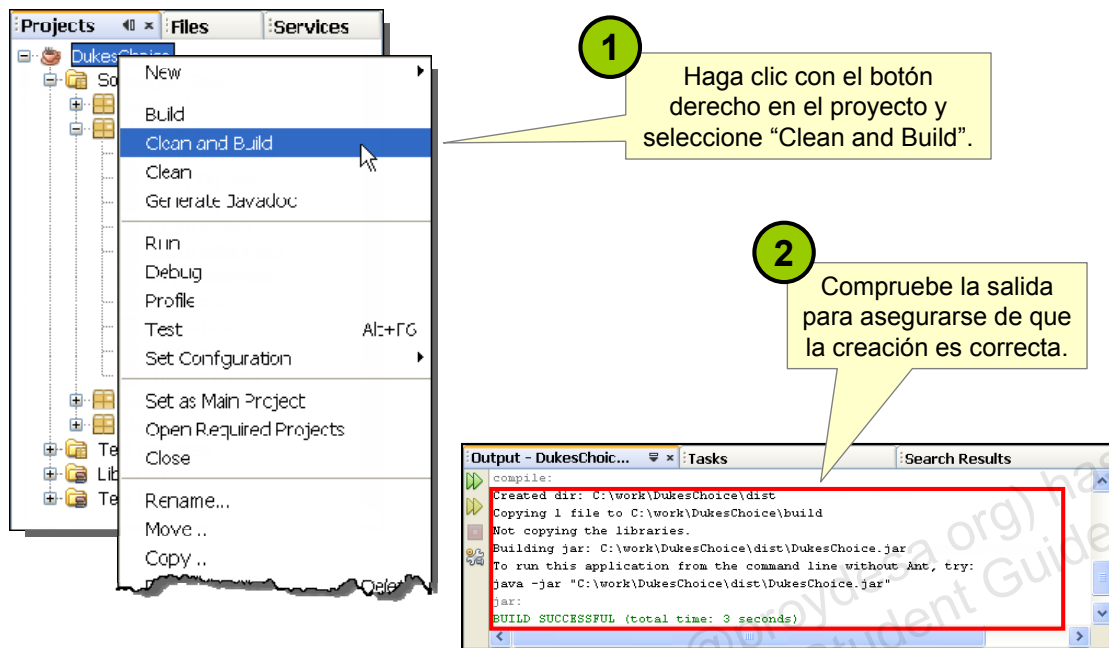


ORACLE

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

Antes de crear el archivo JAR, debe indicar cuál es el archivo que contiene el método `main()`. Éste se escribe posteriormente en el archivo `MANIFEST.MF`.

Creación del archivo JAR con NetBeans



ORACLE

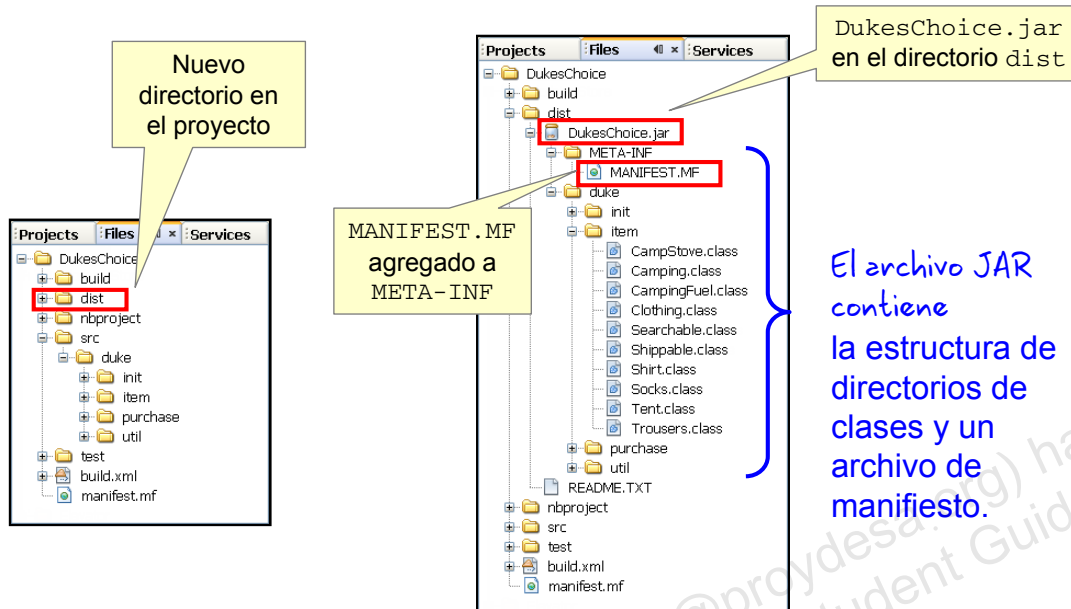
Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Para crear el archivo JAR, haga clic con el botón derecho en el proyecto y seleccione "Clean and Build". Para pequeños proyectos como DukesChoice, esto debe tardar sólo unos segundos.

- Clean elimina cualquier versión anterior.
- Build crea un nuevo archivo JAR.

También puede ejecutar "Clean" y "Build" por separado.

Creación del archivo JAR con NetBeans



ORACLE

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

Por defecto, el archivo JAR se colocará en el directorio dist. (Este directorio se elimina en el proceso de limpieza y se vuelve a crear durante la creación). Al utilizar el separador Files de NetBeans, puede consultar el archivo JAR y asegurarse de que se han agregado las clases correctas.

Temas

- Paquetes
- JAR y despliegue
- **Arquitectura de dos y tres niveles**
- Aplicación Duke's Choice
- Mejoras y modificaciones de la aplicación

ORACLE

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

Arquitectura de cliente/servidor de dos niveles

El sistema de cliente/servidor implica dos o más computadoras que comparten tareas:

- Cada computadora realiza una lógica adecuada a su diseño y función definida.
- El cliente frontend se comunica con la base de datos backend.
- El cliente solicita datos del backend.
- El servidor devuelve los resultados adecuados.
- El cliente maneja y muestra los datos.

ORACLE

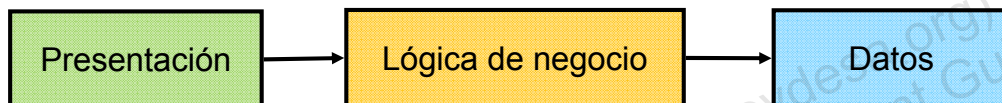
Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

En el cliente/servidor de dos niveles se produce una penalización de rendimiento importante. El software de cliente resulta ser de mayor tamaño y más complejo ya que la mayoría de la lógica se maneja aquí. El uso de la lógica de servidor está limitada a operaciones de base de datos. Aquí se hace referencia al cliente como un *cliente grueso*.

Los clientes gruesos suelen producir un tráfico de red frecuente para el acceso a bases de datos remotas. Esto funciona correctamente para topologías de red basadas en intranet y en la red de área local (LAN), pero produce una huella de gran tamaño en el escritorio en lo relativo a requisitos de memoria y disco. Igualmente, no todos los servidores de base de datos backend son los mismos en cuanto a la lógica de servidor ofrecida y todos ellos tienen sus propios juegos de API que los programadores deben utilizar para optimizar y escalar el rendimiento.

Arquitectura de cliente/servidor de tres niveles

- El cliente/servidor de tres niveles es un enfoque más flexible y complejo.
- Cada nivel se puede sustituir por una implantación diferente:
 - La presentación puede ser mediante GUI, web, smartphone o incluso consola.
 - La lógica de negocio define las reglas de negocio.
 - El nivel de datos es una encapsulación de todos los orígenes de datos existentes.

**ORACLE**

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

Los tres componentes o niveles de un entorno de cliente/servidor de tres niveles son *presentación*, *lógica de negocio o funcionalidad* y *datos*. Están separados para que el software de cualquiera de los niveles se pueda sustituir por una implantación diferente sin que afecte a los otros niveles.

Por ejemplo, si desea sustituir una o varias pantallas orientadas a caracteres con una GUI (nivel de presentación), escribirá la GUI con una interfaz o API establecida para acceder a los mismos programas de funcionalidad de las pantallas orientadas a caracteres.

La lógica de negocio ofrece funcionalidad en cuanto a la definición de todas las reglas de negocio mediante las que se pueden manipular los datos. Los cambios realizados en las políticas de negocio pueden tener un impacto en este nivel sin que ello afecte a las bases de datos reales.

El tercer nivel, o nivel de datos, incluye los sistemas, las aplicaciones y los datos existentes que se han encapsulado para aprovechar esta arquitectura con un esfuerzo mínimo de programación transicional.

Temas

- Paquetes
- JAR y despliegue
- Arquitectura de dos y tres niveles
- **Aplicación Duke's Choice**
- Mejoras y modificaciones de la aplicación

ORACLE®

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

Aplicación Duke's Choice

- Clases abstractas
 - Clothing
 - Ampliada por Shirt y otras clases de ropa
 - Camping
 - Ampliada por Tent y otras clases de acampada
- Interfaces
 - Searchable
 - Todos los artículos que se pueden comprar implementan Searchable.
 - Returnable
 - Los artículos que se pueden devolver implementan Returnable.
 - Shippable
 - Los artículos que se pueden enviar implementan Shippable.

ORACLE

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

Se ha creado una versión de la aplicación Duke's Choice para ilustrar la programación orientada a objetos en Java.

Clase Clothing

```
package duke.item;

public abstract class Clothing implements Searchable, Shippable {
    private String sku = "";
    private int itemID = 0; // Default ID for all clothing items
    private String description = "-description required-"; // default
    private char colorCode = 'U'; // Exception if invalid color code?
    private double price = 0.0; // Default price for all items
    private int quantityInStock = 0;

    public Clothing(int itemID, String description, char colorCode,
                    double price, int quantityInStock ) {
        this.itemID = itemID;
        this.description = description;
        this.colorCode = colorCode;
        this.price = price;
        this.quantityInStock = quantityInStock;
        this.sku = "" + itemID + colorCode;
        ... < more code follows > ...
    }
}
```

ORACLE

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

La clase `Clothing` es muy similar a la clase `Shirt` descrita anteriormente en el curso. Sin embargo, para garantizar que existe un código único para cada tipo de artículo, se ha agregado un campo `SKU` (referencia de almacén).

Clase Clothing

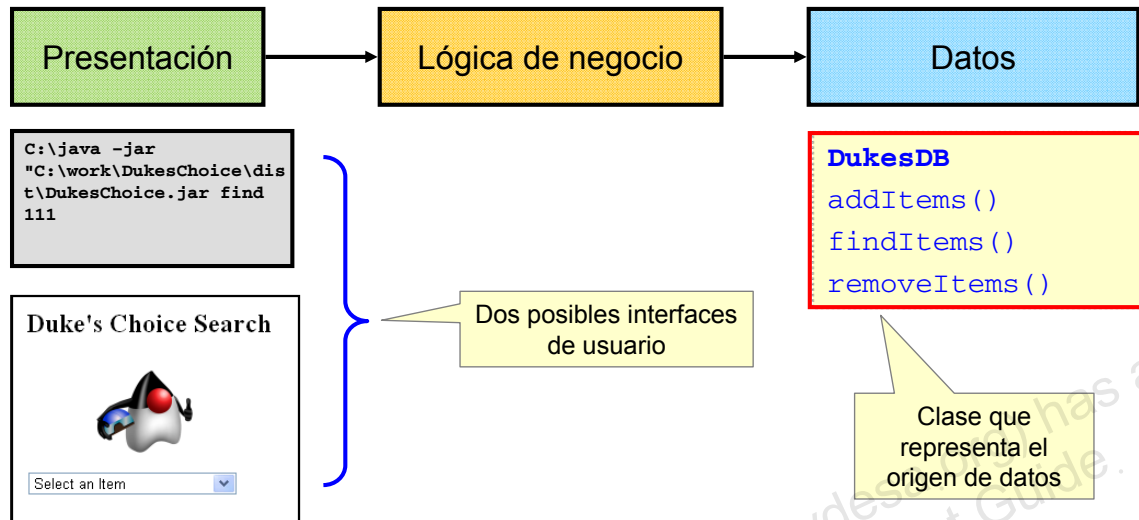
```
public String getDisplay(String separator) {  
  
    String displayString = "SKU: " + getSku() + separator +  
        "Item: " + description + separator +  
        "Price: " + price + separator +  
        "Color: " + colorCode + separator +  
        "Available: " + quantityInStock;  
    return displayString;  
}  
  
... < more code follows > ...
```

ORACLE

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

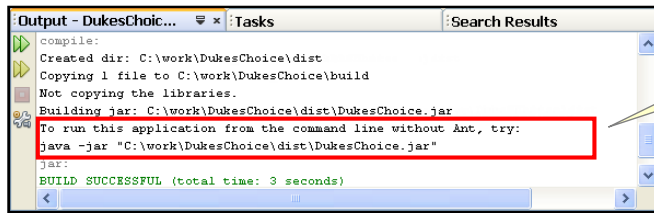
Además del método anterior, `display()`, que se utiliza para mostrar los detalles del artículo, se ha agregado un método `getDisplay()` que devuelve un valor `String`. Esto permite llamar al método a través de diferentes clientes. Toma un argumento: un valor `String` que determina cómo se separan los atributos individuales del artículo. Por ejemplo, se pueden separar con una nueva línea en la versión de la consola de la aplicación o con un elemento HTML para la aplicación web.

Niveles de Duke's Choice

**ORACLE**

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

Ejecución del archivo JAR desde la línea de comandos



Comando que va a ejecutar el archivo JAR

```
C:\java -jar "C:\work\DukesChoice\dist\DukesChoice.jar"
```

Salida:

```
Please add parameters in the format:
  find <item id number>
  OR
  remove <sku> <number to remove>
```

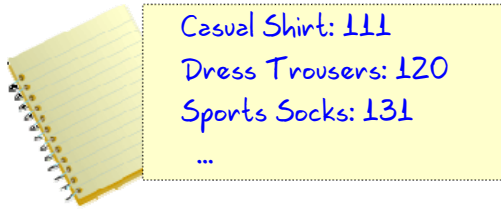
ORACLE

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

La ejecución de la aplicación de línea de comandos con el archivo JAR es muy sencilla y las instrucciones en realidad se proporcionan en la ventana de salida del proceso de creación. (Si se implantaran como una aplicación GUI, se ejecutarían de la misma forma).

Suponiendo que la aplicación es una versión de línea de comandos anterior del software que se ha enviado a Duke's Choice para su prueba, puede ejecutarla tal y como se muestra en la diapositiva. Ya que se trata de una versión anterior, suponga que sólo es para que la utilicen los empleados de Duke's Choice y que es necesario agregar parámetros a la línea de comandos para poder realizar cualquier acción.

Visualización de artículos en la línea de comandos



```
C:\java -jar "C:\work\DukesChoice\dist\DukesChoice.jar find 111
```

Salida:

```
-----
SKU: 111R | Item: Casual Shirt | Price: 34.29 | Color: R | Available: 63
-----
SKU: 111B | Item: Casual Shirt | Price: 25.05 | Color: B | Available: 20
-----
```

ORACLE

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

En esta sencilla aplicación, los comandos se introducen mediante el uso de parámetros de línea de comandos y SKU o ID de artículo. Por lo tanto, puede asumir que se ha proporcionado a los empleados de Duke's Choice una lista de los ID de artículo adecuados para que puedan probar la aplicación.

En el ejemplo, la aplicación busca todos los tipos de camisas de estilo informal que hay en stock. Actualmente, hay dos tipos de camisas de estilo informal en stock: rojas y azules. También puede observar que hay 63 camisas rojas y 20 azules en stock.

Visualización de artículos en la aplicación web de Duke's Choice

The diagram illustrates the user interface of the Duke's Choice Search application across three stages:

- First Screenshot:** Shows the search page with a dropdown menu labeled "Select an Item". A callout states: "La página Search tiene un menú desplegable." (The Search page has a dropdown menu).
- Second Screenshot:** Shows the dropdown menu expanded, listing items: "Casual Shirt", "Select an Item", "Dress Trousers", "Casual Shirt", "Sports Socks", "Dress Socks", "Elite Tent", and "Smokeless camp stove fuel". A callout states: "Aparecen los artículos que están actualmente en stock." (The items currently in stock appear).
- Third Screenshot:** Shows the search results for "Casual Shirt". A table displays the results:

SKU	Description	Price	Available
111R	Casual Shirt	34.29	63
111E	Casual Shirt	25.05	20

 A callout points to the "111E" SKU, stating: "El SKU del artículo es una etiqueta de fijación." (The SKU of the item is a fastener tag). Another callout states: "La selección de un artículo muestra una lista de todos ellos." (The selection of an item shows a list of all of them).

ORACLE

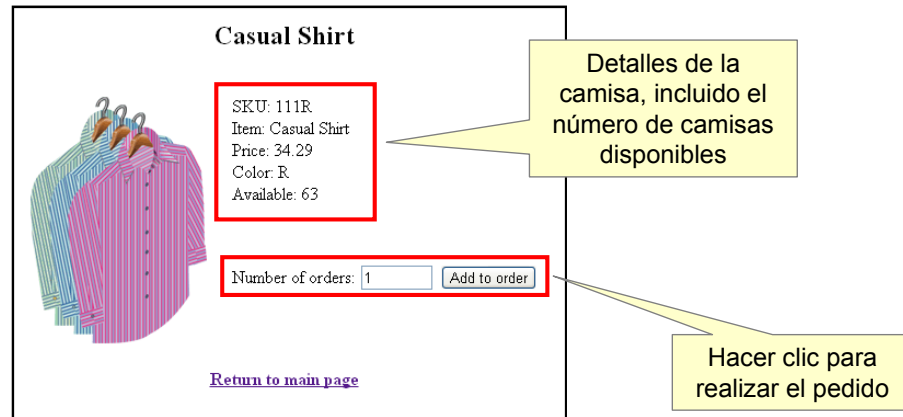
Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

A continuación se describe otra posible aplicación de Duke's Choice: una sencilla aplicación web. En este caso, `DukesChoice.jar` se copia en el servidor de aplicaciones; al que pueden acceder los componentes de la interfaz de usuario de la aplicación (en este caso, a través de los archivos Java Server Pages [JSP]).

La captura de pantalla muestra la página de búsqueda principal que permite a los clientes buscar un artículo concreto. Pueden seleccionar un artículo de una lista desplegable y, a continuación, aparecerán todas las variedades del artículo. En el ejemplo de la diapositiva, la lista muestra la misma información que la aplicación de línea de comandos: camisas de dos colores y la cantidad disponible de cada una.

La aplicación web también permite a un cliente hacer clic en el número de SKU de un artículo concreto. Al hacerlo, accede a la página que muestra más detalles de ese artículo.

Visualización de artículos en la aplicación web de Duke's Choice



ORACLE

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

La captura de pantalla de la diapositiva muestra los detalles del artículo seleccionado por el cliente. En esta página, los clientes pueden agregar un número específico de camisas a sus pedidos.

Las dos aplicaciones mostradas (la aplicación de línea de comandos y esta aplicación web) utilizan clases muy similares a la clase `Shirt` introducida al principio del curso. Aunque la interfaz de usuario de la versión de línea de comandos difiere de la versión web, las clases de artículo (`Shirt`, `Trousers`, `Socks`, `Tent` y `Fuel`) no están implicadas de ninguna forma en la presentación de los datos, por lo que es posible modificar cualquiera de estas clases o agregar clases adicionales sin tener que cambiar la interfaz de usuario.

Temas

- Paquetes
- JAR y despliegue
- Arquitectura de dos y tres niveles
- Aplicación Duke's Choice
- **Mejoras y modificaciones de la aplicación**

ORACLE

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

Mejora de la aplicación

- Un software Java bien diseñado minimiza el tiempo necesario para realizar:
 - Mantenimiento
 - Mejoras
 - Actualizaciones
- Para Duke's Choice, resultaría muy fácil:
 - Agregar nuevos artículos para su venta (lógica de negocio)
 - Desarrollar nuevos clientes (presentación)
 - Instalar la aplicación en un smartphone (por ejemplo)
 - Cambiar el sistema de almacenamiento (datos)

ORACLE

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

En las siguientes diapositivas verá los artículos implicados al agregar otra clase de artículo para representar un traje.

Adición de un nuevo artículo para su venta

Puede agregar un nuevo artículo para su venta mediante:

- La ampliación de la clase Clothing o Camping o incluso la creación de una nueva categoría (por ejemplo, Books)
- La adición de nuevas funciones únicas para el artículo
- La adición de algunos de los nuevos artículos al almacén de datos

ORACLE

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

Adición de un nuevo artículo para su venta

Returnable es una interfaz y se debe implantar.

```

8
9
10
11
12
13
14
public class Suit extends Clothing implements Returnable {
}

```

Suit es un tipo de Clothing.

Las devoluciones están permitidas.

ORACLE

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

NetBeans resulta útil al ampliar las clases abstractas e implantar interfaces, ya que le proporciona indicaciones sobre qué es lo que necesita hacer. En el ejemplo de la diapositiva, la nueva clase `Suit` amplía `Clothing` e implanta `Returnable`. NetBeans indica que debe implantar los métodos de la interfaz `Returnable` (en este caso, el método `doReturn()`).

Implantación de Returnable

```
public class Suit extends Clothing implements Returnable {  
    public String doReturn() {  
        // In the current implementation Returnable provides  
        // a marker that the item can be returned and also returns  
        // a String with conditions for returning the item  
        return "Suit returns must be within 3 days";  
    }  
}
```

ORACLE

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

El código muestra un ejemplo sencillo de la implantación del método `doReturn()` de la interfaz `Returnable`.

Implantación de constructor

```
public class Suit extends Clothing implements Returnable {  
  
    ...< code omitted > ...  
  
    // Types are D = Double-breasted, S = Single-breasted, U=Unset  
    private char suitType = 'U'; //  
  
    // Constructor  
    public Suit(int itemID, String description, char colorCode,  
                double price, char type, int quantityInStock) {  
        super(itemID, description, colorCode, price, quantityInStock);  
        setSuitType(type);  
        setSku(getSku() + type); // To create a unique SKU  
    }  
}
```

ORACLE

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

Este código muestra la implantación del constructor para el tipo Suit, dado que:

- Suit tiene un atributo adicional, suitType, que no está incluido en la superclase Clothing.
- Este atributo adicional se combina con el SKU (generado en la superclase Clothing) para crear un SKU único para este artículo.

Clase Suit: Sustitución de `getDisplay()`

```
public String getDisplay(String separator) {

    String displayString = "SKU: " + getSku() + separator +
        "Item: " + getDescription() + separator +
        "Color: " + getColorCode() + separator +
        "Type: " + getSuitType() + separator +
        "Price: " + getPrice() + separator +

        "Available: " + getQuantityInStock();
    return displayString;
}
```

ORACLE

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

La clase `Clothing` tiene un método `getDisplay(String separator)` en el que se puede especificar un separador para que los atributos del artículo se puedan escribir en una línea y se separen con un carácter separador, o bien se escriban línea por línea mediante el uso de una nueva línea como carácter separador.

El código de la diapositiva muestra la sustitución de `getDisplay(String separator)` para incluir el tipo de traje en la pantalla.

```
C:\>java -jar "C:\work\Java_fundamentals\DukesChoice\dist\DukesChoice.jar" find
410
```

```
-----
SKU: 410BD | Item: Suit | Color: B | Type: D | Price: 999.99 | Available: 21
```

```
-----
SKU: 410BS | Item: Suit | Color: B | Type: S | Price: 789.99 | Available: 15
```

```
-----
SKU: 410gD | Item: Suit | Color: G | Type: D | Price: 999.99 | Available: 21
```

```
-----
SKU: 410WS | Item: Suit | Color: W | Type: S | Price: 789.99 | Available: 15
-----
```

Implantación de los métodos getter y setter

```
public class Suit extends Clothing implements Returnable {

    ...< code omitted > ...

    public char getSuitType() {
        return suitType;
    }

    public void setSuitType(char suitType) {
        if (suitType!='D' && suitType!='B') {
            throw new IllegalArgumentException("The suit type must be"
                                           + " either D = Double-breasted "
                                           + "or S = Single-breasted");
        }
        this.suitType = suitType;
    }
}
```

ORACLE

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

El código muestra la implantación de los métodos getter y setter para el tipo de traje. Si 'D' o 'B' no se transfiere al constructor, el método devuelve una excepción `IllegalArgumentException`. Observe que `IllegalArgumentException` es una excepción no comprobada, por lo que no es necesario que se devuelva desde este método o se compruebe en el método de llamada.

Suponiendo que no se obtiene en la implantación actual de la aplicación, si se transfiere un argumento no válido al método, los responsables de probar la aplicación Duke's Choice verán lo siguiente:

```
C:\>java -jar
```

```
"C:\work\Java_fundamentals\DukesChoice\dist\DukesChoice.jar"
```

```
find 410
```

```
Exception in thread "main" java.lang.IllegalArgumentException:
```

```
The suit type must be either D = Double-breasted or S = Single-breasted
```

```
at duke.item.Suit.setSuitType(Suit.java:43)
```

```
at duke.item.Suit.<init>(Suit.java:20)
```

```
at duke.init.DukesDB.setupDb(DukesDB.java:52)
```

```
at duke.init.DukesDB.<init>(DukesDB.java:84)
```

```
at duke.init.DBtest.main(DBtest.java:29)
```

Actualización de aplicaciones con la clase Suit

Para la aplicación de línea de comandos:

- Cree un nuevo archivo `DukesChoice.jar`.
- (Opcional) Cópielo en una nueva ubicación del sistema de archivos o en otra máquina.

Para la aplicación web:

- Cree un nuevo archivo `DukesChoice.jar`.
- Cópielo en el directorio que utiliza el servidor de aplicaciones para los archivos de biblioteca.

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.

Observe que el archivo JAR es exactamente el mismo en ambos casos.

Prueba de la clase Suit: Línea de comandos

```
C:\>java -jar
      "C:\work\Java_fundamentals\DukesChoice\dist\DukesChoice.jar"
      find 410
```

```
-----
SKU: 410BD | Item: Suit | Price: 999.99 | Color: B | Available: 21
```

```
-----
SKU: 410BS | Item: Suit | Price: 789.99 | Color: B | Available: 15
```

```
-----
SKU: 410gD | Item: Suit | Price: 999.99 | Color: G | Available: 14
```

```
-----
SKU: 410WS | Item: Suit | Price: 789.99 | Color: W | Available: 18
-----
```



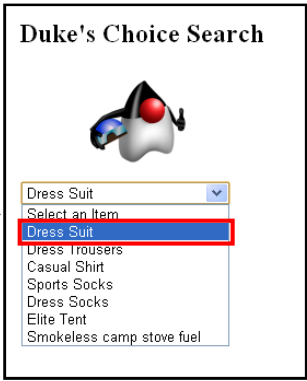
Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

Ahora los responsables de pruebas de Duke's Choice pueden buscar los trajes que hay en stock. Sin embargo, la pantalla no les permite saber si el traje es de botonadura cruzada o sin cruzar.

Prueba de la clase Suit: Aplicación web

Aparece un nuevo artículo en el menú desplegable.

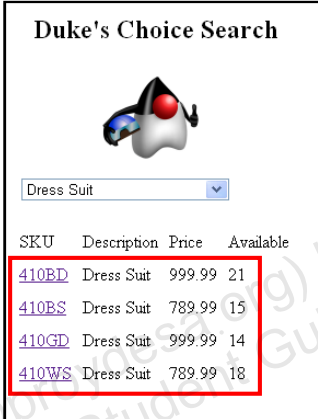
Duke's Choice Search



Select an Item
Dress Suit
Dress Trousers
Casual Shirt
Sports Socks
Dress Socks
Elite Tent
Smokeless camp stove fuel

Se muestran los diferentes tipos de trajes agregados al almacén de datos.

Duke's Choice Search



SKU	Description	Price	Available
410ED	Dress Suit	999.99	21
410BS	Dress Suit	789.99	15
410GD	Dress Suit	999.99	14
410WS	Dress Suit	789.99	18

ORACLE

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

Después de reiniciar la aplicación web, los responsables de pruebas pueden ver un artículo adicional en el menú desplegable para Dress Suit, además de los distintos tipos de Dress Suit que se han agregado al almacén de datos con sus SKU.

Adición de la clase Suit a la aplicación web

**ORACLE**

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

Al hacer clic en uno de los artículos Dress Suit que aparecen, se mostrarán los detalles. Observe que, ya que se ha sustituido el método `getDisplay()`, aparecerá el tipo de traje (S para botonadura sin cruzar). No se han realizado modificaciones en la aplicación web.

Resumen

En esta lección, debe haber aprendido lo siguiente:

- Desplegar una aplicación simple como un archivo JAR
- Describir las partes de una aplicación Java, lo que incluye la interfaz de usuario y el backend
- Describir cómo se pueden ampliar las clases para implantar nuevas capacidades en la aplicación



ORACLE

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

Sin prácticas para esta lección

Esta lección no tiene prácticas.

ORACLE

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

Resumen del curso

En este curso, debe haber aprendido lo siguiente:

- 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 en 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.

Resumen 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.

Resumen 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.

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

Referencia rápida de 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.

1. Declarar una clase

```
public class Shirt{ ←declaración de clase
}
```

2. Declarar un campo/variable

```
public char colorCode; ←variable de campo
int counter; ←variable local
```

3. Declarar e inicializar una variable primitiva

```
public double price = 0.0; ←variable de campo
int hour = 12; ←variable local
```

4. Declarar e instanciar una referencia de objeto

```
public ArrayList names = new ArrayList();
```

5. Llamar a un método

```
displayInformation(); ←método sin argumentos ni valor de retorno
setColorCode('R'); ←método con un argumento y sin valor de retorno
int level = getLevel(); ←método sin argumentos pero que devuelve un valor
```

6. Declarar un método

```
public void displayInformation(){...} ←método: sin argumentos, devuelve un valor nulo
public String getName() {...} ←método: sin argumentos, devuelve String
public void setName(String name){...} ←método: argumento String, devuelve un valor nulo
```

7. Bloque If/else

```
If (name1.equals(name2)) {
    System.out.println();
}
else {
    System.out.println("Different name.");
}
```

8. Construcción Switch ←Sintaxis

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

9. Estructura de una clase

```

package myClasses; ←sentencia package
import java.util.ArrayList; ←sentencia import

public class NamesList{ ←declaración de clase
    public ArrayList names = new ArrayList(); ←campo

    public void setList(){ ←método
        // code_block;
    } ←fin de método
} ←fin de clase

```

10. Construcción while ←sintaxis

```

while (boolean_expression) {
    // realizar esto mientras que la expresión sea true
    // code_block;
} // fin del bloque while

```

11. Construcción do/while ←sintaxis

```

do {    // realizar lo siguiente una vez antes de evaluar la expresión

    // a continuación, realizar esto mientras que la expresión sea
    true

    // code_block
}
while (boolean_expression);

```

12. Bucle for ←sintaxis

```

for (data_type init_var; boolean_expression; increment){
    // code_block;
}

```

←ejemplo

```

for (int i = 1; i<10; i++){
    System.out.println("Array element: " + myArray[i]);
}

```

13. Bucle for mejorado ←sintaxis

```
for (data_type var : array_name ) {  
    // code_block;  
}
```

←ejemplo

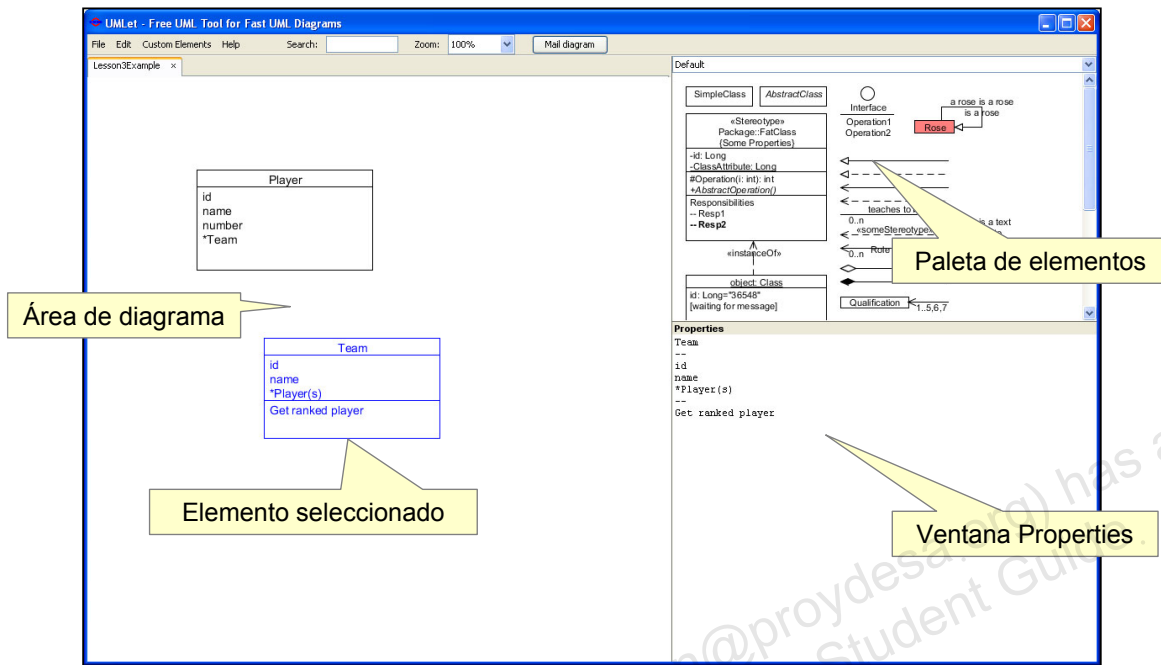
```
for (Object obj : myList){  
    System.out.println("List element: "+ obj);  
}
```


Consejos para UMLet

ORACLE®

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

Interfaz por defecto de UML



ORACLE

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

1. Adición de elementos al diagrama

Haga clic dos veces en cualquier elemento de la paleta; aparecerá en la esquina superior izquierda de la ventana principal del diagrama.

2. Duplicación de elementos en el diagrama

Haga clic dos veces en un elemento para duplicarlo. También puede copiar y pegar (o utilizar sus respectivos equivalentes de teclado Ctrl + C y Ctrl + V).

3. Selección de varios elementos

Mantenga pulsada la tecla Ctrl para seleccionar varios elementos.

4. Selección de lazo de varios elementos

Pulse Ctrl y haga clic para seleccionar un rectángulo que contenga los elementos que desee.

5. Cambio de los elementos UML

Seleccione un elemento y modifique sus atributos en el panel de texto de la parte inferior derecha. Cada tipo de elemento tiene un lenguaje de marcado sencillo (por ejemplo, el texto `/ClassName/` convierte `ClassName` a cursiva). Los lenguajes de marcado se explorarán mejor mediante los elementos UML de ejemplo de las paletas.

6. Introducción de comentarios en una descripción de elemento UML

UMLet soporta comentarios de estilo C++. Al empezar una línea con `/"` (por ejemplo, `/"my comment.."`) permite a UMLet ignorar esa línea de marcado.

7. Cambio del color de los elementos UML

Haga clic con el botón derecho en un elemento y seleccione el color de primer o segundo plano mediante el menú contextual.

8. También puede introducir simplemente el nombre del color en la descripción del elemento (por ejemplo, `"bg=black"` o `"fg=red"`).

9. Creación de relaciones UML

Haga clic dos veces en una relación y arrastre sus puntos finales hasta los bordes de los elementos UML, se quedarán anclados en esta ubicación.

10. Edición de las relaciones

Muchas herramientas UML pueden tardar algún tiempo en cambiar el tipo o la dirección de una relación. En UMLet, simplemente modifique el tipo de línea (es decir, cambie la línea `"lt="` en la descripción del elemento). Por ejemplo, cambie `"lt=<"` a `"lt=->"` para cambiar la dirección, el tipo de flecha y los puntos de la línea al mismo tiempo.

11. Etiquetado de las relaciones

Edite el nombre de una relación en la descripción de la relación.

Los nombres de rol se pueden especificar con `"r1="` o `"r2="`.

Para multiplicidades, utilice `"m1="` o `"m2="`.

Los cualificadores se crean con `"q1="` o `"q2="`.

12. Creación de diagramas de secuencia

Cambie la paleta actual a `"Sequence - all in one"`. Puede agregar el elemento de diagrama de secuencia al diagrama si hace clic dos veces en él.

Este lenguaje de marcado del elemento es un poco más complejo. La idea principal es que cada carril tenga un nombre y un ID (definido por la cadena `"_name~ID_"`). Los ID se pueden utilizar para definir mensajes entre carriles (por ejemplo, `"id1->id3"`).

13. Creación de diagramas de actividad

Cambie la paleta actual a `"Activity - all in one"`. Puede agregar el elemento del diagrama de actividad al diagrama si hace clic dos veces en él.

Aquí, los separadores de la descripción del elemento se utilizarán para definir las bifurcaciones de la actividad.

Conceptos básicos de UML

Unified Modeling Language (UML) es un lenguaje gráfico para modelar sistemas de software. UML no es:

- Un lenguaje de programación: es un juego de diagramas que se puede utilizar para especificar, construir, visualizar y documentar diseños de software. Los ingenieros de software utilizan los diagramas UML para construir y explicar sus diseños de software, al igual que los arquitectos utilizan los planos para construir y explicar sus diseños de construcción. UML tiene diagramas que le ayudarán en todas las fases del desarrollo de aplicaciones, desde la recopilación de requisitos hasta el diseño, la codificación, la prueba y el despliegue.
- Un proceso para el análisis y el diseño: sus diagramas se deben utilizar con un proceso.

UML fue desarrollado a principios de los 90 por tres líderes del mundo del modelado de objetos: Grady Booch, James Rumbaugh e Ivar Jacobson. Su objetivo era unificar los métodos principales que anteriormente habían desarrollado para crear un nuevo estándar para el modelado de software. UML es ahora el lenguaje de modelado más utilizado. La especificación de UML la mantiene actualmente el grupo Object Management Group (OMG) y está disponible en el sitio web de OMG, en <http://www.omg.org/uml/>.

Elementos generales

En general, los diagramas de UML representan:

- Conceptos, que se describen como símbolos (también denominados *nodos*).
- Relaciones entre estos conceptos, que se describen como rutas de acceso (también denominados *enlaces*) que conectan los símbolos.

Estos nodos y enlaces están especializados para cada diagrama concreto. Por ejemplo, en los diagramas de clase, los nodos representan las clases de objeto y los enlaces representan las relaciones entre clases y relaciones de generalización (herencia).



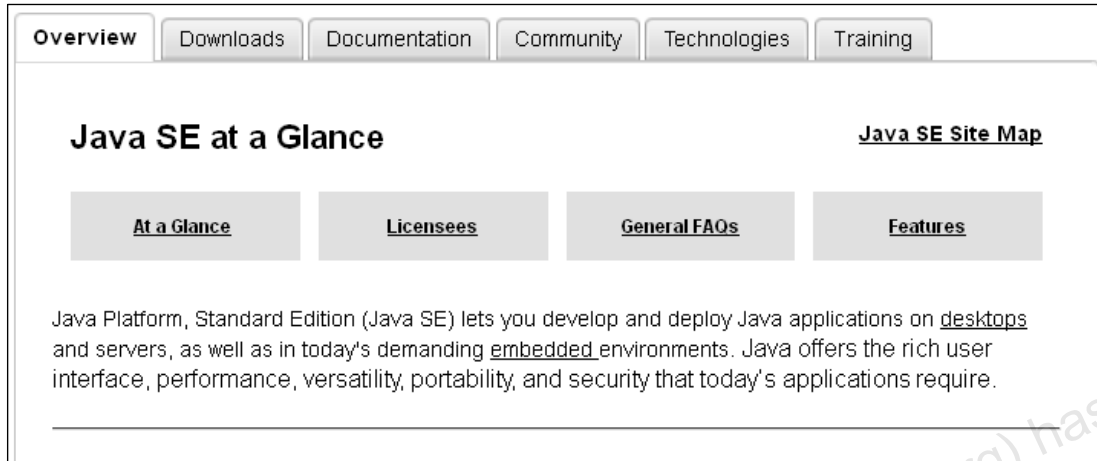
Recursos

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.

Java en Oracle Technology Network (OTN)



Overview Downloads Documentation Community Technologies Training

Java SE at a Glance

[Java SE Site Map](#)

[At a Glance](#) [Licensees](#) [General FAQs](#) [Features](#)

Java Platform, Standard Edition (Java SE) lets you develop and deploy Java applications on desktops and servers, as well as in today's demanding embedded environments. Java offers the rich user interface, performance, versatility, portability, and security that today's applications require.

ORACLE

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

Puede encontrar varios recursos en las páginas de Java SE 7 de OTN, entre los que se incluyen:

- Descargas
- Documentación
- Comunidad Java
- Tecnologías
- Formación

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Descargas de Java SE

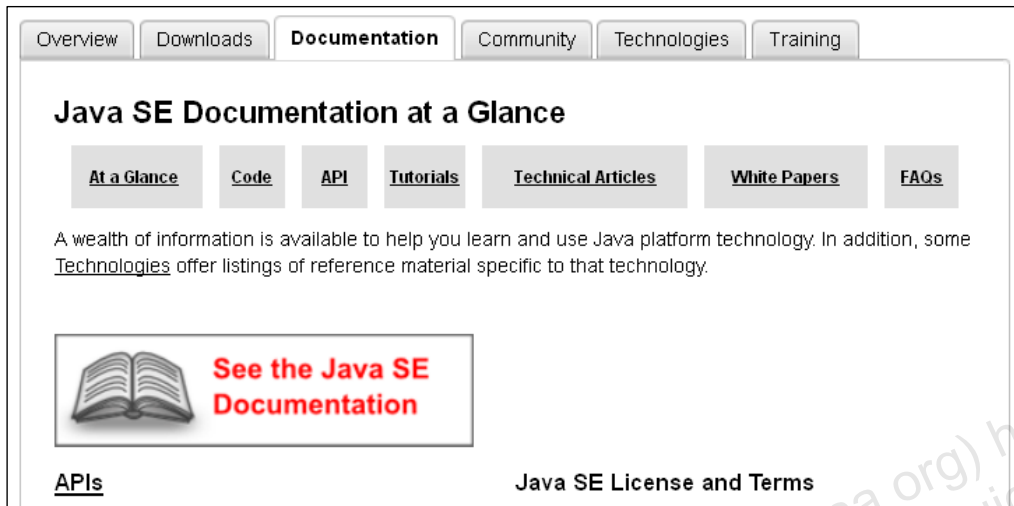


Copyright © 2011, Oracle y/o sus filiales. Todos los derechos reservados.

El enlace Downloads proporciona las versiones más recientes y las anteriores de Java SE (en tiempo de ejecución y JDK), JavaFX, Java EE y NetBeans.

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Documentación de Java



ORACLE

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

Puede encontrar varios recursos en la página Documentation, entre los que se incluyen:

- Código
- API
- Tutoriales
- Artículos técnicos
- ...y mucho más

El enlace Java SE Documentation incluye información adicional del desarrollador como:

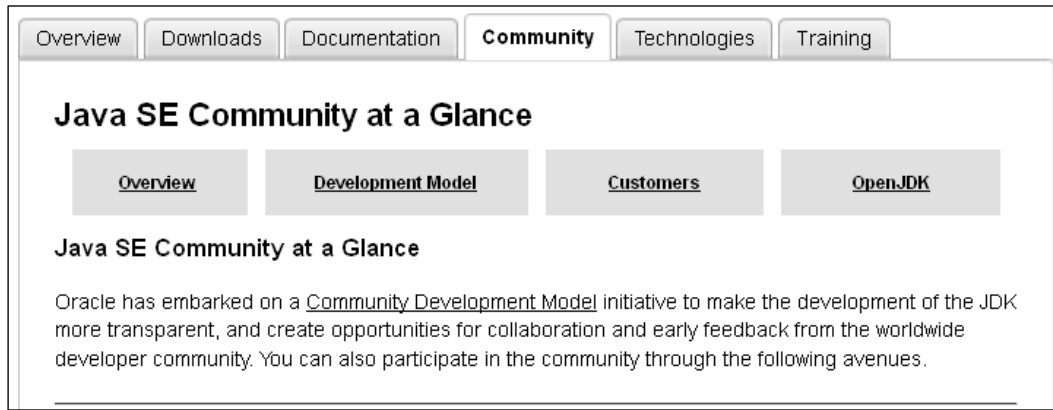
- Documentación de API
- Lenguaje Java y especificaciones sobre las máquinas virtuales
- Guías del desarrollador
- Instrucciones de instalación de JDK/JRE
- ...y mucho más

Página Documentation:

<http://www.oracle.com/technetwork/java/javase/documentation/index.html>

Página Java SE Technical Documentation: <http://download.oracle.com/javase/>

Comunidad Java



ORACLE

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

¿Qué es la comunidad Java? Con frecuencia, oímos hablar de la comunidad Java y de una variedad de acrónimos relacionados con Java con los que puede que no esté familiarizado, como JUG, JCP EC y OpenJDK.

En general, la comunidad Java es el término que se utiliza para hacer referencia a los numerosos miembros y organizaciones que desarrollan, innovan y utilizan la tecnología Java.

La página Java Community incluye enlaces a:

- **Foros:** los foros de discusión de la tecnología Java son paneles de mensajes interactivos para compartir ideas y puntos de vista sobre tecnologías Java y técnicas de programación.
- **Grupos de usuarios:** los miembros de los grupos de usuarios de Java se reúnen con frecuencia para intercambiar información e ideas técnicas.
- **Java Developer Newsletter:** Java Developer Newsletter es una publicación en línea gratuita y mensual que incluye noticias, artículos técnicos y eventos.
- **Blogs** como los siguientes:
 - The Java Source
 - Blogs de Oracle Java
- **Eventos del desarrollador de Java**

<http://www.oracle.com/technetwork/java/javase/community/index.html>

Comunidad Java: Enfoque extensivo



ORACLE

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

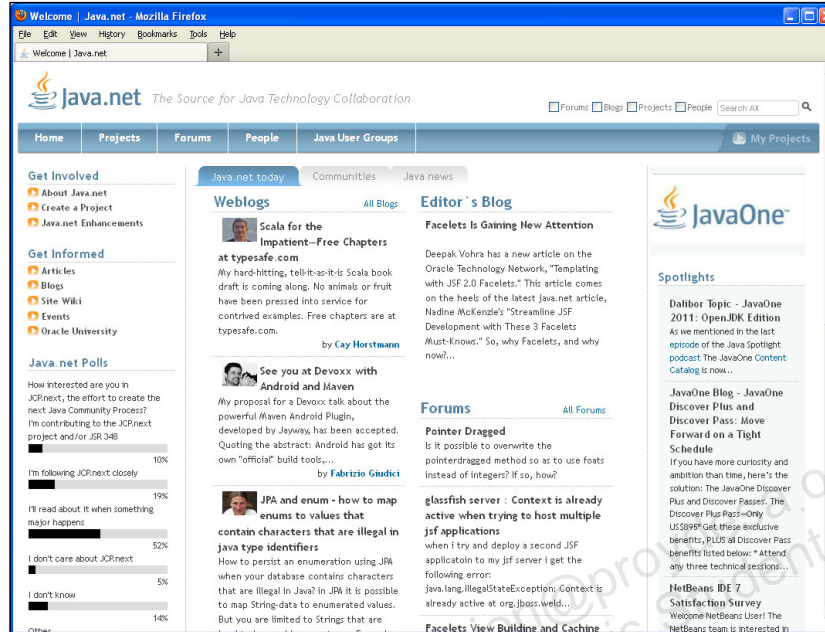
Foros: los foros de discusión de la tecnología Java son paneles de mensajes interactivos para JUG.

Un grupo de usuarios de Java (JUG) es un grupo de personas que comparten intereses comunes en la tecnología Java y se reúnen con frecuencia para compartir información e ideas técnicas. La estructura real de un JUG puede variar considerablemente: desde un número pequeño de amigos y compañeros que se reúnen de manera informal alguna tarde hasta un gran grupo de compañías situadas en la misma área geográfica. Independientemente del tamaño y el enfoque de un JUG concreto, el sentido del espíritu de grupo es el mismo.

OpenJDK (también se conoce como **Open Java Development Kit**): implantación gratuita y de código abierto del lenguaje de programación Java. Además de Oracle, otros contribuyentes como RedHat, IBM y Apple colaboran con OpenJDK.

JCP: JCP significa Java Community Process, un proceso formalizado que permite a las partes interesadas implicarse en la definición de futuras versiones y funciones de la plataforma Java. El JCP Executive Committee (EC) es el grupo de miembros que guía la evolución de la tecnología Java. EC representa a los participantes y representantes de sección más importantes de la comunidad Java.

Comunidad Java: Java.net

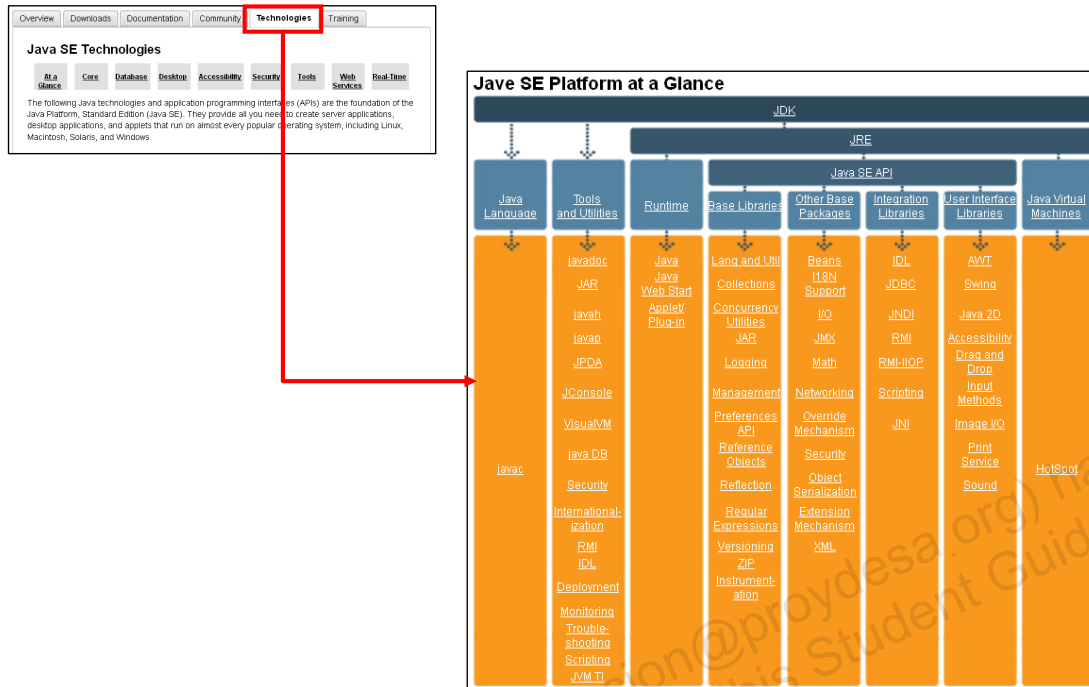


ORACLE®

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

Java.net es una gran comunidad de desarrolladores de Java y sus proyectos. Está abierta a que cualquier persona que esté interesada en Java, las tecnologías relacionadas con JVM y la formación visite los debates y proyectos del sitio. Java.net gestiona los proyectos de una forma diferente a la de la mayoría de los grupos con el mantenimiento de las comunidades de proyectos. Es decir, los proyectos que utilizan tecnologías similares o que son del mismo tipo se agrupan en un área para que sea más fácil encontrar a otros desarrolladores con intereses y conocimientos similares así como sus proyectos. El sitio proporciona artículos técnicos, noticias sobre eventos y blogs.

Tecnologías Java



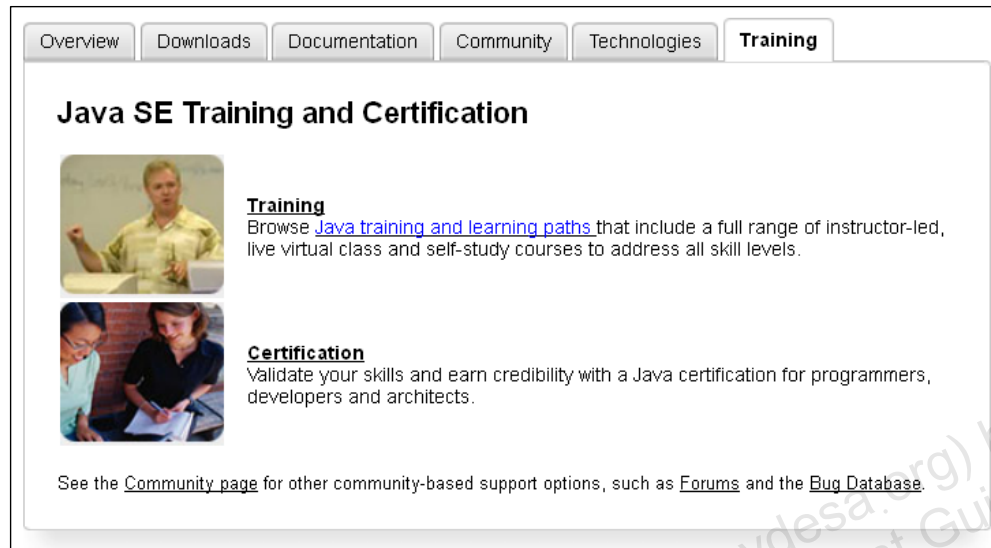
ORACLE®

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

La página Java Technologies incluye un mapa de clics que describe todas las tecnologías de la plataforma Java SE detalladamente.

<http://www.oracle.com/technetwork/java/javase/tech/index.html>

Formación de Java



ORACLE

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

La página “Java SE Training and Certification” describe la formación Java disponible, así como el programa de certificación Java. Oracle University ofrece cursos que le ayudarán a conocer la tecnología y el lenguaje de programación Java para que pueda generar código de forma más inteligente y desarrollar programas y aplicaciones robustos con más rapidez y con cualquier plataforma, incluidos el servidor de aplicaciones de Oracle y el software de la infraestructura web. Certifique sus conocimientos y dedicación con la certificación Java, una de las credenciales más reconocidas del sector.

Los cursos de formación de Java SE más recientes incluyen:

- *Java SE 7 New Features*
- *Java Performance Tuning and Optimization*
- *Conceptos fundamentales de Java SE 7*
- *Java SE 7 Programming*

<http://www.oracle.com/technetwork/java/javase/training/index.html>

Oracle Learning Library

ORACLE Learning Library Login

Home **Advanced Search** Bookmarks My Reviews About

Home > Advanced Search

Search content title, description and tags...

Product Family **Java** Product Click icon to select -> Sub Product Click icon to select ->

Release Click icon to select -> Content Type Click icon to select -> Release Date Click icon to select ->

Tag Click icon to select -> Rating Click icon to select -> **Reset** **Query**

Title	Type	Release Date	Duration	Rating	Tags
Installing a MIDP Application on a Real Device	Video	17-Jul-11	6 mins	★★★★★	Java, Java ME, OracleLearnin
NetBeans IDE 7.0 Overview	Video	24-May-11	9 mins	★★★★★	GlassFish, Java, Java SE, Ja NetBeans, OracleLearning, Y
Using JLayer in Swing Applications	Video	20-May-11	4 mins	★★★★★	JLayer, Java, Java SE, Orac Swing, Youtube
Creating Your Own Application Using the Java ME SDK	Video	08-May-11	5.25 mins	★★★★★	Java, Java ME, OracleLearnit
Downloading and Installing the Java Micro Edition SDK	Video	03-May-11	2.5 mins	★★★★★	Java, Java ME, OracleLearnit
Running a Java Micro Edition Sample Application	Video	03-May-11	4 mins	★★★★★	Java, Java ME, OracleLearnit
Rich Applications for Billions of Devices: What's New in LWJGL?	Article	04-Apr-11		★★★★★	Java, Java ME

ORACLE

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

Las funciones de Oracle Learning Library (OLL) incluyen artículos técnicos, documentación técnica, vídeos, demostraciones y tutoriales de Oracle by Example (OBE) sobre varios temas, incluido Java. El sitio necesita una conexión a Oracle Technology Network (OTN), pero todo el contenido es gratuito. OLL está disponible en oracle.com/oll.

Java Magazine



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

Suscríbase a *Java Magazine*, una revista bimensual que es una fuente de conocimientos esencial sobre la tecnología Java, el lenguaje de programación Java y las aplicaciones basadas en Java para personas que basan en esta tecnología sus carreras profesionales o que aspiran a ello.

<http://www.oracle.com/technetwork/java/javamagazine/index.html>

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