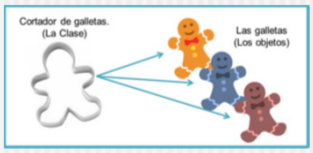
Unidad 8
Clases y objetos en Java

Programación
1º D.A.M.

1

Contenido

1. Creación y destrucción de objetos
2. Acceso a los miembros de una clase
3. Modificadores *static* y *final*
4. Arrays de objetos
5. Herencia
6. Polimorfismo
7. Clases abstractas
8. Clases internas
9. Interfaces
10. Relaciones todo – parte



2

1. Creación y destrucción de objetos

1. Instanciación de objetos
   1. Operador new
   2. Constructores
2. Destrucción de objetos

3

1.1. Instanciación de objetos. Operador *new*

- Definición de una clase

```
class Clase
{
   tipo1 atributo1;
   ...
   tipoN atributoN;

   tipo metodo1(parametros)
   { // Cuerpo del método; }
   tipo metodo2(parametros)
   { // Cuerpo del método; }
}
```

4

1.1. Instanciación de objetos. Operador *new*

- Creación de referencia para objeto de una clase

```
<nombre_clase> nombreObjeto;
```

- Reserva de espacio para el objeto

```
nombreObjeto = new <nombre_clase>();
```

- En un único paso

```
<nombre_clase> nombreObjeto =
        new <nombre_clase>();
```

5

1.1. Instanciación de objetos. Constructores

- Constructor
  - Define cómo iniciar objeto (atributos privados)
- Características
  - Nunca devuelve un valor en su declaración
  - Devuelve referencia a instancia de la clase
  - Rellena los atributos de valores iniciales
  - Invocado con el operador `new`

6

1.1. Instanciación de objetos. Constructores

- Tipos
  - Constructores con parámetros
    - Creados por el usuario
  - Constructor por defecto
    - No lleva parámetros
    - Inicia números a `0` y referencias a `null`
    - Creado por compilador, si no existe ningún constructo
- Referencia `this`
  - Apunta a la instancia que llama a un método

7

1.2. Destrucción de objetos

- No existe operador *delete*
- Destrucción automática
  - *Garbage collector*
    - Recolector de basura
    - Cuando objeto no tiene referencia activa
    - Se libera automáticamente su memoria
  - Inconveniente
    - Sobrecarga en tiempo de ejecución
  - Ventaja
    - Libera al programador de la tarea

8

2. Acceso a los miembros de una clase

1. Paquetes
2. Especificadores de acceso
   1. public
   2. private
   3. protected
   4. friendly

9

2.1. Paquetes

- Características
  - Permiten organizar aplicaciones en directorios
  - Agrupan bibliotecas de clases
  - Estructura jerárquica
  - Restricciones de acceso y visibilidad
  - Nombre de clase
    - `nombrePaquete.nombreClase`
  - Paquete por defecto
    - Cuando no se define paquete
    - `public` se aplicará a clases del mismo directorio

10

2.1. Paquetes

- Declaración
  - Palabra reservada `package`

```
package mamiferos;
class Ballena{
   ...
}
```

```
package animales.mamiferos;
class Ballena{
   ...
}
```

11

2.1. Paquetes

- Acceso a otros paquetes
  - Palabra reservada `import`

```
import nombre_paquete;

import animales.mamiferos.*;
```

12

2.1. Paquetes

- Algunos paquetes de Java
  - java.applet
    - AppletContext, AppletStub, AudioClip
  - java.awt
    - Button, Checkbox, Choice, Menu, Panel, TextArea, ..
  - java.io
    - FileInputStream, FileOutputStream, FileReader, ...
  - java.lang
    - Object,Exception,System,Integer,Math,String,Package, ...
  - java.sql
    - Array, Connection, Driver, ResultSet, SQLData, Date, ...
  - java.util
    - Date, List, Map, Random, Stack, Map, Dictionary, ...

13

2.2. Especificadores de acceso

- Palabra que antecede a declaración de
  - Clase
  - Método
  - Atributo
- Determinan alcance de visibilidad del elemento
  - Modificadores
    - `public`
    - `protected`
    - `private`
  - Si no se usa modificador
    - friendly

14

2.2. Especificadores de acceso

- `public`
  - Miembros de uso público
  - Accesibles desde fuera de la definición
    - Acceso desde métodos de la propia clase
    - Acceso desde métodos de otras clases
  - Para mantener encapsulación
    - Atributos (propiedades) → private
    - Métodos (comportamiento) → public

15

2.2. Especificadores de acceso

- `private`
  - Miembros de uso privado
  - Accesibles sólo desde la clase
    - Acceso desde métodos de la propia clase
  - Para mantener encapsulación
    - Atributos (propiedades) → `private`
    - Métodos (comportamiento) → `public`

16

2.2. Especificadores de acceso

- `protected`
  - Miembros de uso protegido
  - Accesibles desde la clase y heredadas
  - Accesibles desde otra clase del mismo paquete
  - No accesibles desde clases de otros paquetes
    - A no ser que sean heredadas
- `friendly`
  - Modificador por defecto
  - Accesibles desde clases del mismo paquete

17

2.2. Especificadores de acceso

Zona	private (privado)	friendly (Sin modificador)	protected (protegido)	public (público)
Misma clase	X	X	X	X
Subclase en mismo paquete		X	X	X
Clase en el mismo paquete		X	X	X
Subclase en otro paquete			X	X
No subclase en otro paquete				X

18

3. Modificadores *static* y *final*

19

3. Modificadores *static* y *final*

- Objetivo de cada modificador
  - `final` →Para miembros constantes
  - `static` →Para miembros de clase
- Usándolos adecuadamente se consiguen
  - Constantes de clase y objeto
  - Atributos y métodos con alcance de clase

20

3. Modificadores *static* y *final*

- Combinaciones posibles
  - `static final`
    - Constantes de clase, asignados a la iniciación
    - Comunes a todos los objetos de la clase
    - Invocables sin ninguna instancia
  - `final`
    - Valores constantes
    - Potencialmente distintos en distintas instancias
    - Valor inicial en constructor, y no modificable
  - `static`
    - Valores comunes para todas las instancias
    - Potencialmente variables
  - Los demás
    - Diferentes en cada objeto de la clase

21

4. Arrays de objetos

22

4. Arrays de objetos

- Definición
  - `NombreClase[] objetos;`
  - `NombreClase objetos[];`
- Instanciación del array
  - `objetos = new NombreClase[N];`
- Definición + Instanciación
  - `NombreClase[] objetos = new NombreClase[N];`
  - `NombreClase objetos[] = new NombreClase[N];`
- Instanciación de objetos del array
  - `objetos[i] = new NombreClase();`
- Tamaño del array
  - Atributo público `length`
  - `objetos.length`

23

5. Herencia

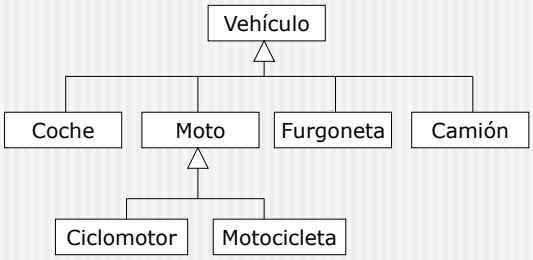
24

5. Herencia

- Creación de nuevas clases
  - Con características comunes a otras
    - Propiedades (atributos)
    - Comportamiento (métodos)
  - Con características particulares
- Evita tener que crear clases desde 0
  - Herencia de atributos y métodos comunes
  - Definición de atributos y métodos propios

25

5. Herencia



Jerarquía con herencia simple

26

5. Herencia

- Sintaxis

```
class <claseDerivada> extends <claseBase>
{
   <calificador> <tipo> <nuevoAtributo1>;
   ...
   <calificador> <tipo> <nuevoAtributoN>;
   <calificador> <tipo> <nuevoMetodo1>(params)
   {
      ...
   }
   ...
   <calificador> <tipo> <nuevoMetodoM>(params)
   {
      ...
   }
}
```

27

5. Herencia

- Constructores invocados
  - Constructor clase base
    - Inicia parte correspondiente a clase base
    - Invocación con palabra reservada `super`
  - Constructor clase derivada
    - Inicia parte de la clase derivada

28

5. Herencia

- Orden de invocación de constructores
  - Si no se instancian objetos en la definición de atributos
    1. Constructor de clase base
    2. Código de constructor de clase derivada
       - Cuando aparezca *new* en dicho código...
         - Constructor de la clase correspondiente
  - Si se instancian objetos en la definición de atributos
    - Constructor de clase base
    - Constructores de atributos instanciados
    - Código de constructor de clase derivada
      - Cuando aparezca `new` en dicho código...
        - Constructor de la clase correspondiente

29

5. Herencia

- Métodos que se heredan
  - Métodos heredados
    - Heredan definición e implementación
    - Se usan igual en clase base y derivada
    - Ej. : `calcularEdad()` en `Persona` y `Empleado`
  - Métodos sobreescritos o reimplementados
    - Heredan definición, no implementación
    - Necesitan otra implementación en clase derivada
      - Implementación completamente nueva
      - Implementación de la base, y más instrucciones
        - `super.metodo()`
    - Ej. : `mostrarAtributos()` en `Persona` y `Empleado`

30

5. Herencia

```
class <derivada> extends <base>
{
   void visualizar()
   {
      super.visualizar();
      ...
   }
   ...
}
```

31

6. Polimorfismo

32

6. Polimorfismo

- Clases de diferentes tipos referenciadas por la misma variable
  - Definición de objeto de la clase base
    - Ej. : Definir objeto `Figura`
  - Uso del objeto como de la clase derivada
    - Ej. : Usar objeto como `Circulo`, `Rectangulo`, ...
  - Sobreescribir métodos de la clase base
    - Se usará el método que corresponda
    - Ej. : Método `perimetro()` adecuado a la figura

33

7. Clases abstractas

34

7. Clases abstractas

- Métodos abstractos
  - Métodos teóricos
  - No se desean implementar en la clase
  - Sólo aparecen indicados
  - Implementados en las clases derivadas
    - El desarrollador está obligado a ello
    - Sobreescritura del método
  - Modificador `abstract`

```
abstract public double area();
```

35

7. Clases abstractas

- Clases abstractas
  - Incluyen métodos abstractos
  - Creadas para ser heredadas por otras
    - Clases base para herencia
  - No pueden ser instanciadas
  - Modificador `abstract`

```
abstract class clase {
    tipo atributo;
    ...
    abstract [acceso][tipo] metodoAbstracto([args]);
    [acceso] [tipo] metodoConcreto([args])
    {
      ...
    }
}
```

36

8. Clases internas

37

8. Clases internas

- Definidas dentro de otra clase
- La clase definida dentro será muy dependiente
- Usadas para objetos internos a una clase
  - Pasan a ser atributos de la clase
- Pueden ser privadas, protegidas y públicas
- Fuera de la clase contenedora no se pueden crear objetos de la contenida
  - Salvo que la contenida sea `static`
    - Poco sentido

38

8. Clases internas

```
public class Coche {
   public int velocidad;
   public Motor motor;
   // Aquí métodos del Coche
   // Clase interna
   public class Motor {
      // Definición de la clase Motor
   }
}
```

39

9. Interfaces

40

9. Interfaces

- Conjunto de constantes y métodos abstractos
  - Métodos
    - Abstractos (no se precisa el modificador `abstract`)
    - Públicos
    - Meros prototipos
  - Atributos
    - Han de ser iniciados
    - Se consideran constantes
- Definen comportamiento de objetos
- Implementadas por clases
  - Deben implementar todos los métodos
- Una interfaz puede heredarse de otra interfaz

41

9. Interfaces

- Creación de interfaces

```
interface Arrancable
{
   boolean arrancado = false;
   void arrancar();
   void detener();
}
```

42

9. Interfaces

- Uso de interfaces

```
class Coche extends Vehiculo implements Arrancable {

   public void arrancar()
   {
      ...
   }

   public void detener()
   {
      ...
   }
}
```

43

9. Interfaces

- Herencia de interfaces
  - Permiten herencia múltiple
    - A diferencia de lo que ocurre con las clases
  - Superinterfaz / Subinterfaz

```
interface dibujable extends
  escribible, pintable {
   ...
}
```

44

10. Relaciones todo – parte

1. Composición
2. Agregación

45

10. Relaciones todo – parte

- Todo: elemento "contenedor"
  - Por ejemplo `Perro`
- Parte: elemento "contenido"
  - Por ejemplo `Hocico`
  - Por ejemplo `Pulga`

46

10. Relaciones todo – parte

- Tipos de relación todo – parte

1. Composición
   - Vida del contenido ligada a la del contenedor
   - Por ejemplo `Perro – Hocico`
2. Agregación
   - Vida del contenido desligada del contenedor
   - Por ejemplo `Perro – Pulga`

47

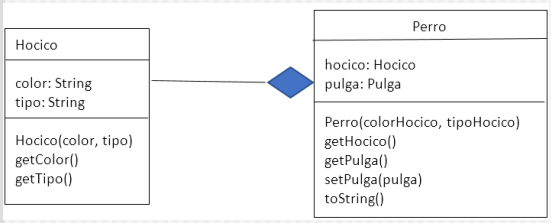
10.1. Relación de composición

- Características
  - Vida del contenido ligada a la del contenedor
    - Creación – Vida – Destrucción
  - El contenido no puede existir sin el contenedor

48

10.1. Relación de composición

- Representación en UML
  - Rombo relleno hacia el contenedor



49

10.1. Relación de composición

- Implementación en Java
  - Instancia contenido con el contenedor

```
public class Perro {

   private Hocico hocico;
   private Pulga pulga;

   Perro(String colorHocico, String tipoHocico){
      hocico = new Hocico(colorHocico, tipoHocico);
   }
```

50

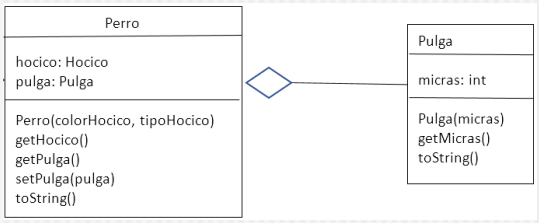
10.2. Relación de agregación

- Características
  - Vida del contenido desligada del contenedor
    - Creación – Vida – Destrucción independientes
  - El contenido SÍ puede existir sin el contenedor

51

10.2. Relación de agregación

- Representación en UML
  - Rombo SIN relleno hacia el contenedor



52

10.2. Relación de agregación

- Implementación en Java
  - Contenido creado fuera
    - Asociado después al contenedor

```
public class Perro {

    private Hocico hocico;
    private Pulga pulga;

    Perro(String colorHocico, String tipoHocico){
        hocico = new Hocico(colorHocico, tipoHocico);
    }

    public void setPulga(Pulga pulga) {
        this.pulga = pulga;
    }
```

53

Unidad 8
Clases y objetos en Java

Programación
1º D.A.M.

54