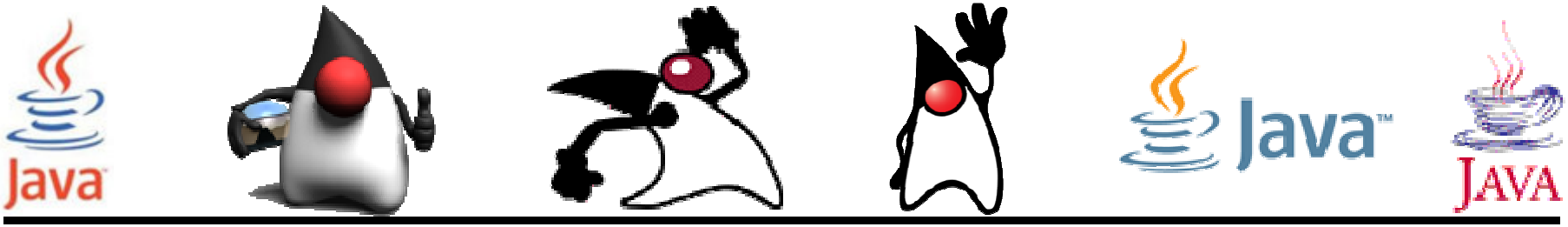


Herencia

Pedro Corcuera

Dpto. Matemática Aplicada y
Ciencias de la Computación

Universidad de Cantabria

corcuerp@unican.es

Objetivos

- Aprender los conceptos de herencia
- Comprender la forma de derivar una subclase

Índice

- ¿Qué es herencia?
- La Clase Object
- Cómo derivar una subclase
- Qué se puede hacer en una subclase?
- Palabra reservada super
- Sobreescritura (overriding) de métodos
- Modificadores en los métodos sobreescritos
- Casting de clases
- Clases y métodos final

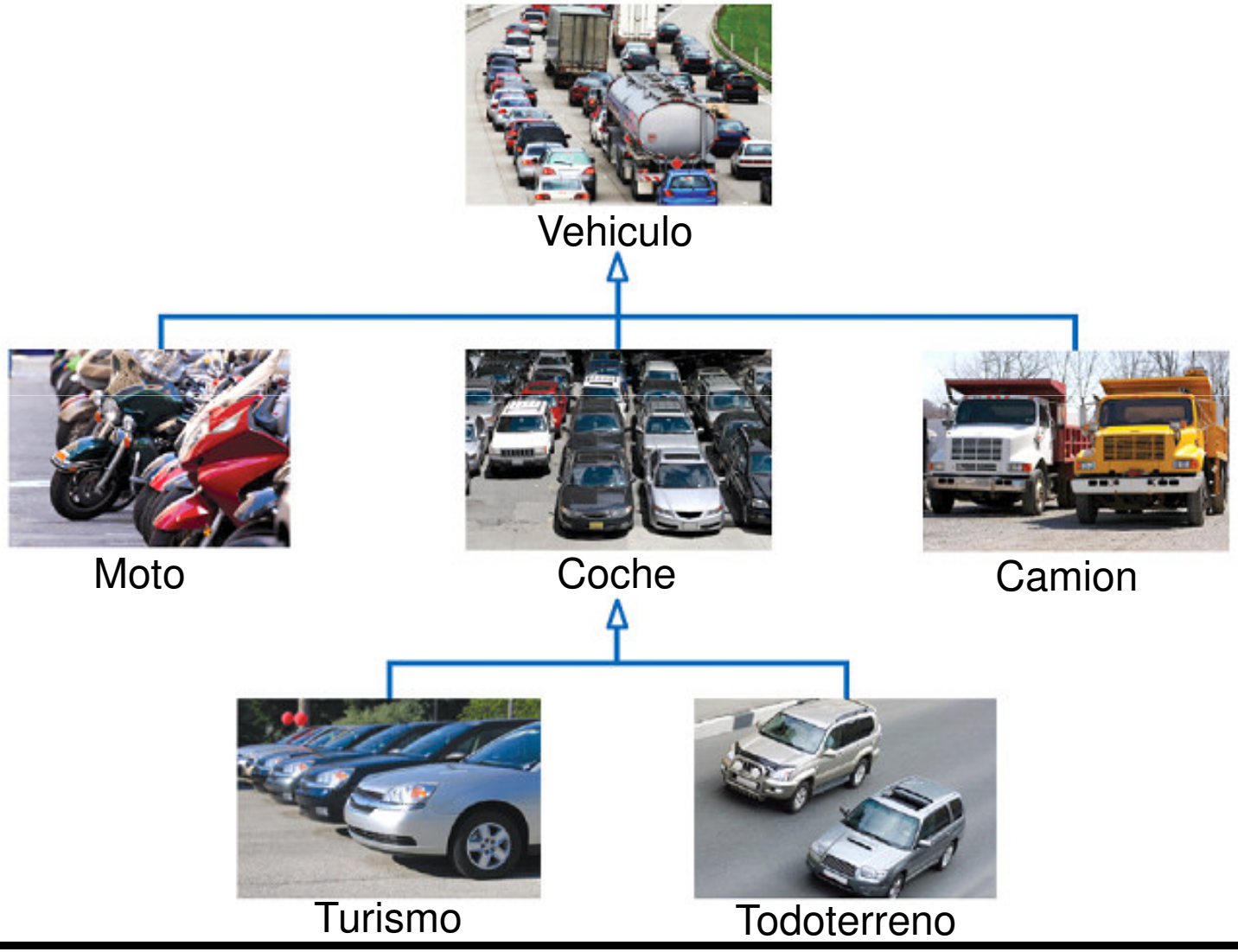
¿Qué es herencia?

- Característica primaria de la programación orientada a objetos
- La herencia es una relación entre
  - una *superclase*: clase más general
  - una *subclase*: clase más especializada
- La subclase 'hereda' los datos (variables) y comportamientos (métodos) de la superclase

¿Porqué la herencia? Reusabilidad

- Un comportamiento (método) en una super clase es heredado por todas las subclases.
  - Se escribe un método una vez y se puede utilizar por todas las subclases
- Un conjunto de propiedades (campos) en una super clase son heredadas por todas las subclases.
  - Una clase y todas las subclases comparten el mismo conjunto de propiedades
- Una subclase sólo necesita implementar las diferencias entre ella y su antecesor

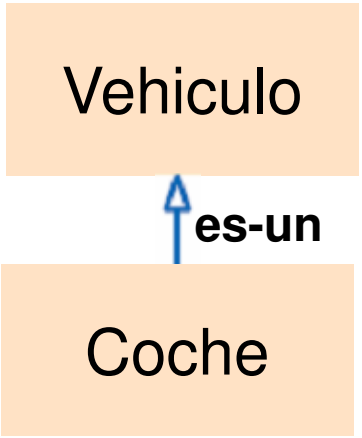
Ejemplo: Jerarquía de Clase Vehículo



Principio de sustitución

- Como la subclase Coche "**es-un**" Vehiculo
  - Coche comparte características comunes con Vehiculo
  - Se puede sustituir un objeto Coche en un algoritmo que espera un objeto Vehiculo
    - La relación es-un se representa con una flecha en un diagrama de clase y significa que la subclase puede pertenecer como un objeto de la superclase

```
Coche miCoche = new Coche(. . .);
procesaVehiculo(miCoche);
```



Herencia de las superclases

- Una subclase hereda desde la superclase:
  - Todos los métodos públicos que no sobreescribe (override)
  - Todas las variables de instancia
- La subclase puede:
  - Añadir variables de instancia nuevas
  - Añadir nuevos métodos
  - Cambiar la implementación de métodos heredados

La Clase Object

- La clase Object es la madre de todas las subclases
  - En lenguaje Java, todas las clases son derivadas o extendidas a partir de la super clase Object
  - La clase Object es la única que no tiene una clase padre
- La clase Object define e implementa el comportamiento común a todas las clases incluyendo las que uno escribe
  - getClass()
  - equals()
  - toString() …

Llamada a método toString

- El método `toString` retorna la representación String de un objeto
- La clase Rectangle (java.awt) tiene un método toString
  - Se puede invocar directamente

```
Rectangle box = new Rectangle(5, 10, 20, 30);
String s = box.toString();   // Invoca toString directamente
// Asigna s a "java.awt.Rectangle[x=5,y=10,width=20,height=30]"
```

  - Se invoca implicítamente cuando se concatena un String con un objeto

```
System.out.println("box=" + box); // Llamada toString implicito
```

Sobreescribiendo el método toString

- Para una clase creada se puede usar `toString` sin sobreescribir

```
CuentaBanco miAhorro = new CuentaBanco(5000);
String s = miAhorro.toString();
// Asigna a s algo como “CuentaBanco@d24606bf"
```

  - Contiene el nombre de la clase seguido del código hash del objeto (no es muy útil)

- Si se sobreescribe, se puede obtener el estado del objeto

```
public class CuentaBanco {
  public String toString() {
// retorna “CuentaBanco[balance=5000]"
    return "CuentaBanco[balance=" + balance + "]";}  }
```

Herencia y el método toString

- Es preferible usar getclass (heredado del objeto) en la superclase

```
public class CuentaBanco { . . .
  public String toString()
  {
    return getClass().getName() + "[balance=" + balance + "]";
  }
```

  - Después usar herencia, llamando primero toString de la sc

```
public class CuentaAhorro extends CuentaBanco {
  . . .
  public String toString()
  {
    return super.toString() + "[interestRate=" + intRate + "]";
  }  // retorna CuentaAhorro[balance= 10000][interestRate= 5]
}
```

Cómo derivar una subclase

- Para derivar una clase hija se usa la palabra reservada `extends`
- Suponiendo que se tiene una clase padre Person

```
public class Person {
  protected String name;
  protected String address;
  /** Default constructor */
  public Person() {
    System.out.println("Inside Person:Constructor");
    name = ""; address = "";
  }
  . . . .
}
```

Cómo derivar una subclase

- Se desea crear otra clase llamada Student
- Como un estudiante también es una persona, se decide extender la clase Person que hereda todas las propiedades y métodos de la clases existente Person

```
public class Student extends Person {
  public Student(){
    System.out.println("Inside Student:Constructor");
  }
  . . . .
}
```

Qué se puede hacer en una subclase?

- Una subclase hereda todos los miembros (campos y métodos) public y protected de su padre, sin importar el package en el que se encuentra la subclase
- Si la subclase está en el mismo paquete del padre, también hereda los miembros privados del paquete (campos o métodos) del padre

Qué se puede hacer en una subclase respecto a los campos

- Los campos heredados se pueden usar de forma directa, como cualquier otro campo
- Se puede declarar nuevos campos en la subclase que no están en la superclase
- Se puede declarar un campo en la subclase con el mismo nombre como el de la superclase, por lo que se le oculta (no recomendado)
- Una subclase no hereda los miembros privados de la clase padre. La sc debe ofrecer métodos públicos o protegidos para acceder a los campos privados

Qué se puede hacer en una subclase respecto a los métodos

- Los métodos heredados se pueden usar de forma directa
- Se puede escribir una nueva instancia del método en la subclase que tiene la misma interfaz del método en la superclase (overriding)
- Se puede escribir un nuevo método estático en la subclase que tiene la misma interfaz como el de la superclase (ocultación)
- Se puede declarar nuevos métodos en la subclase que no están en la superclase

Llamada encadenada de constructor

- Un constructor de una subclase invoca el constructor de la superclase implícitamente
  - Cuando un objeto Student, una subclase, es instanciado el constructor por defecto de su superclase, clase Person, es invocado implícitamente antes que el método constructor de la subclase es invocado
- El constructor de una subclase puede invocar el constructor de su superclase explícitamente usando la palabra reservada super
  - Se usa para pasar parámetros al constructor de la supercl

Ejemplo: Llamada a constructor sc

- Considerando el siguiente código:

```
public static void main (String [] args) {
   Student ana = new Student();
}
```

- Crea un objeto de la clase Student. La salida del programa es:

```
Inside Person:Constructor
Inside Student:Constructor
```

Palabra reservada super

- Una subclase puede llamar *explícitamente* un constructor de su superclase inmediata
- Esto se consigue por el uso de super para la llamada al constructor
- Una llamada del constructor super() en el constructor de una subclase resultará en la ejecución del constructor de la superclase basado en los argumentos pasados
  - La llamada a super() debe ser la primera sentencia en el constructor y sólo se puede usar en un constructor

Palabra reservada super

- Otro uso de super es para referirse a miembros de la superclase (como la referencia this)
- Ejemplos de uso de super:

```
public class Student extends Person {
  public Student(){
    super("UnNombre", "UnaDireccion");
    System.out.println("Inside Student:Constructor");
    super.name = "UnNombre";
    super.address = "UnaDireccion";
  }
  . . . .
}
```

Sobreescritura (overriding) de métodos

- Si una clase derivada necesita tener una versión diferente de la instancia de un método de la superclase, sobreescribe la instancia del método en la subclase
- El método que sobreescribe tiene el mismo nombre, número y tipo de parámetros y tipo de retorno que el método sobreescrito
- El método que sobreescribe puede también retornar un subtipo del tipo retornado por el método sobreescrito. Al tipo retornado se le llama covariante

Ejemplo de (overriding) de métodos

```
public class Person {
   :
   public String getName() {
     System.out.println("Parent: getName");
     return name;
   }
}
```

```
public class Student extends Person {
   :
   public String getName() {
     System.out.println("Student:getName");
     return name;
   }
   :
}
```

Cuando se invoca el método getName de un objeto Student, el resultado es: `Student:getName`

Modificadores en los métodos sobreescritos

- El especificador de acceso de un método sobreescrito puede permitir más, pero no menos, acceso que el método sobreescrito
  - Así, un método protegido en la superclase se puede hacer público, pero no privado, en la subclase
- Se obtendrá un error en tiempo de compilación si se intenta cambiar una instancia de método en la superclase a un método de clase en la subclase, y viceversa

Ocultación de Métodos

- Si una subclase define un método de clase (método estático) con la misma interfaz que el método de clase en la superclase, el método en la subclase *esconde* al de la superclase

```
class Animal {
  public static void testClassMethod() {
    System.out.println("metodo de clase en Animal.");
  }
}
// testClassMethod() de la subclase oculta el de la superclase
class Cat extends Animal {
  public static void testClassMethod() {
    System.out.println("metodo de clase en Cat.");
  }
}
```

Ocultación de campos

- Dentro de una subclase, un campo que tiene el mismo nombre que un campo en la superclase oculta el campo de la superclase incluso si son de tipos diferentes
- Dentro de la subclase, el campo en la superclase no puede ser referenciado simplemente por su nombre
  - Para ello debe ser accedido con la palabra clave super
- En general, no se recomienda el uso de campos ocultos debido a que dificulta la lectura del código

Qué es “Tipo”?

- Cuando se crea una instancia de un objeto de una clase, se dice que el objeto es del “tipo” de la clase y sus superclases
- Ejemplo:

```
Student student1 = new Student();
```

  - El objeto student1 es del tipo Student
  - El objeto student1 también es del tipo Person si Student es una subclase de Person
  - El objeto student1 también es del Object

Qué es Significado?

- La instancia de un objeto de un tipo particular se puede usar en cualquier lugar donde se invoca una instancia del tipo y del supertipo
- Ejemplo:
  - El objeto student1 es de tipo Student y Person
  - El objeto student1 se puede usar en cualquier lugar donde el objeto de tipo Student o Person se invoca
- Este mecanismo hace posible polimorfismo

Casting de tipo implícito

- Un objeto de una subclase se puede asignar a una variable (referencia) de la clase padre a través del casting de tipo implícito
- Ejemplo:
  - Si la clase Student es hija de la clase Person y UCStudent es hija de la clase Student

```
UCStudent ucstudent = new UCStudent();
Student student = ucstudent; //cast implicito
Person person = ucstudent; //cast implicito
Object object = ucstudent; //cast implicito
```

Casting de tipo explícito

- Un objeto de una superclase se puede asignar a una variable (referencia) de la clase hija a través del casting de tipo explícito.
  - No hacerlo produce error de compilación
- Ejemplo:
  - Si la clase Student es hija de la clase Person

```
Person person1 = new Student();
Student student1=(Student)person1;//cast expl
```

Excepción en ejecución por tipos dispares

- Incluso con cast explícito se puede tener un error en tiempo de ejecución
- Ejemplo:
  - La clase Student es hija de la clase Person
  - La clase Teacher es hija de la clase Person

```
Person person1 = new Student();
Person person2 = new Teacher();
Student student1=(Student)person1;//cast expl
//produce error de type mismatch en ejecución
Student student2=(Student)person2;
```

Uso del operador instanceof para prevenir error por disparidad de tipo

- Se puede comprobar el tipo del objeto usando el operador instanceof antes de realizar el cast
- Ejemplo: la clase Student es hija de la clase Person y la clase Teacher es hija de la clase Person

```
Person person1 = new Student();
Person person2 = new Teacher();
//Hacer cast despues de verificar tipo
if (pearson2 instanceof Student) {
  Student student2=(Student)person2;
}
```

Clases final

- Son clases que no se pueden extender. Declaración:

```
public final ClassName {
...
}
```

- Ejemplos de clases finales son las clases wrapper y la clase String

Métodos final

- Son métodos que no se pueden sobreescribir.
  Declaración:

```
public final
[tipoReturn][nombreMet]([parametros]) {
 ...
 }
```

- Los métodos static son automáticamente final

```
 public final String getName() {
  return name;
 }
```