Programación de Computadores: Java - Programación Orientada a Objetos

Juan F. Pérez

Departamento MACC
Matemáticas Aplicadas y Ciencias de la Computación
Universidad del Rosario

juanferna.perez@urosario.edu.co

Segundo Semestre de 2017

Contenidos

Clases y Objetos

Clases y Objetos

En programación orientada a objetos:

Clases y Objetos

En programación orientada a objetos:

- Definimos clases de objetos

Clases y Objetos

En programación orientada a objetos:

- Definimos clases de objetos
- Creamos objetos (instancias) de clases ya definidas

Clases y Objetos

En programación orientada a objetos:

- Definimos clases de objetos
- Creamos objetos (instancias) de clases ya definidas
- En Java cada clase se define en una archivo `nombre.java`

Un objeto es una instancia (caso) de una clase de objetos

Clases

- Encabezado:

```
public class nombre extends superClase{
}
```

Clases

- Encabezado:

```
public class nombre extends superClase{
}
```

- *public*: visible a todas las clases en el programa

Clases

- Encabezado:

```
public class nombre extends superClase{
}
```

- *public*: visible a todas las clases en el programa
- *class*: definimos una clase

Clases

- Encabezado:

```
public class nombre extends superClase{
}
```

- *public*: visible a todas las clases en el programa
- *class*: definimos una clase
- *nombre*: nombre de la clase

Clases

- Encabezado:

```
public class nombre extends superClase{
}
```

- *public*: visible a todas las clases en el programa
- *class*: definimos una clase
- *nombre*: nombre de la clase
- *extends*: (opcional) la clase extiende una clase existente

Clases

- Encabezado:

```
public class nombre extends superClase{
}
```

- *public*: visible a todas las clases en el programa
- *class*: definimos una clase
- *nombre*: nombre de la clase
- *extends*: (opcional) la clase extiende una clase existente
- *superClase*: (opcional) la clase existente que extiende la nueva clase (súper-clase)

Clases y Super-clases

- Clases definidas como una **jerarquía**

Clases y Super-clases

- Clases definidas como una **jerarquía**
- Nuevas clases (sub-clases) extienden clases existentes (súper-clases)

Clases y Super-clases

- Clases definidas como una **jerarquía**
- Nuevas clases (sub-clases) extienden clases existentes (súper-clases)
- Toda clase es una sub-clase de la clase `Object`

Clases y Super-clases

- Clases definidas como una **jerarquía**
- Nuevas clases (sub-clases) extienden clases existentes (súper-clases)
- Toda clase es una sub-clase de la clase `Object`
- **Herencia** (inheritance):
  La sub-clase *hereda* toda la funcionalidad de la súper-clase

Clases

Una clase está compuesta por las siguientes entradas:

- **Variables de instancia** (atributos o campos):
  - Definen el *estado* de la instancia

Clases

Una clase está compuesta por las siguientes entradas:

- **Variables de instancia** (atributos o campos):
  - Definen el *estado* de la instancia
- **Constantes**

Clases

Una clase está compuesta por las siguientes entradas:

- **Variables de instancia** (atributos o campos):
  - Definen el *estado* de la instancia
- **Constantes**
- **Constructores**:
  - Permiten *crear* instancias (objetos) de esta clase

Clases

Una clase está compuesta por las siguientes entradas:

- **Variables de instancia** (atributos o campos):
  - Definen el *estado* de la instancia
- **Constantes**
- **Constructores**:
  - Permiten *crear* instancias (objetos) de esta clase
- **Métodos**:
  - Definen el *comportamiento* de la instancia

Visibilidad de las entradas

Opciones de visibilidad:

- **public**: visible a todas las clases en el programa

Visibilidad de las entradas

Opciones de visibilidad:

- **public**: visible a todas las clases en el programa
- **private**: visible solo dentro de la clase donde está definida

Visibilidad de las entradas

Opciones de visibilidad:

- **public**: visible a todas las clases en el programa
- **private**: visible solo dentro de la clase donde está definida
- **protected**: visible solo a clases y subclases en el mismo paquete

Visibilidad de las entradas

Opciones de visibilidad:

- **public**: visible a todas las clases en el programa
- **private**: visible solo dentro de la clase donde está definida
- **protected**: visible solo a clases y subclases en el mismo paquete
- : *package-private*, i.e., visible solo a clases en el mismo paquete

Encapsulación

Un objeto o instancia:

- Actúa como una entidad que puede manipularse

Encapsulación

Un objeto o instancia:

- Actúa como una entidad que puede manipularse
- A pesar de estar compuesto por múltiples entradas

Encapsulación

Un objeto o instancia:

- Actúa como una entidad que puede manipularse
- A pesar de estar compuesto por múltiples entradas
- Define una barrera que permite limitar el acceso de entidades extrañas al objeto mismo

Encapsulación

Un objeto o instancia:

- Actúa como una entidad que puede manipularse
- A pesar de estar compuesto por múltiples entradas
- Define una barrera que permite limitar el acceso de entidades extrañas al objeto mismo
- Por defecto queremos que las entradas de una clase sean privadas, a menos que sea necesaria una mayor visibilidad

Ejemplos de Clases Existentes

Clases Existentes

- ¿Cómo sabemos la funcionalidad de una clase?
- ¿Qué métodos podemos usar?
- ¿Qué parámetros reciben esos métodos?
- ¿Qué retornan esos métodos?

Clases Existentes

- ¿Cómo sabemos la funcionalidad de una clase?
- ¿Qué métodos podemos usar?
- ¿Qué parámetros reciben esos métodos?
- ¿Qué retornan esos métodos?

Documentación: `javadoc`

Ejemplo (librería `acm`):

`http://cs.stanford.edu/people/eroberts/jtf/javadoc/student/index.html`

Librería `acm`

- Lista de Paquetes
- Lista de Clases
- Clase `ConsoleProgram`
  - Jerarquía de clases
  - Campos (variables de instancia)
  - Constructores
  - Métodos: funcionalidad, parámetros, retorno
  - Métodos heredados

Otra clase: `acm.util.RandomGenerator`

- Generador de números aleatorios
- Constructor especial:

```
private RandomGenerator rgen = RandomGenerator.
    getInstance();
```

- **Note el . para llamar el método** `getInstance()`
- Un método que usaremos:

```
int     nextInt(int low, int high)
```

- Queremos *simular* lanzamientos de un dado

Usando acm.util.RandomGenerator

```
package randomNumbers;
import acm.program.ConsoleProgram;
import acm.util.RandomGenerator;
public class LanzarDados extends ConsoleProgram {
 public void run (){
  RandomGenerator ranGen = RandomGenerator.getInstance();
  int numRep = readInt ("␣Ingrese␣␣el␣número␣de␣veces␣que␣
    quiere␣lanzar␣el␣dado\n␣");
  int suma = 0;
  for(int i = 0; i < numRep; i++){
   int resDado = ranGen.nextInt(1, 6);
   suma = suma + resDado;
   println ("El␣␣lanzamiento␣número␣␣"+ (i+1) +"␣es␣"+
    resDado );
  }
  println ("La␣suma␣de␣todos␣los␣lanzamientos␣es␣"+ suma);
 }
}
```

Definiendo nuevas clases

Clase `Estudiante`

Queremos una clase que:

- Represente a un estudiante de la universidad
- La clase debe representar el estado de una estudiante, incluyendo:
  - Nombre
  - Número de Documento
  - Créditos aprobados
  - Si está al día en pagos o no
- Abramos la clase `Estudiante.java` (paquete `estudiantes`)

Variables y Constantes

```
private String nombre;

private int numDocumento;

private float creditosAprobados;

private boolean alDiaPagos;

public static final double CREDITOS_PARA_GRADO = 145.0;
```

`static`: variable (constante) disponible al nivel de la clase, no de una instancia en particular

Constructor

```
public Estudiante(String nombre, int numDoc){
 this.nombre = nombre;
 numDocumento = numDoc;
}
```

Mismo nombre que la clase.

Getters y Setters

```
public void setCreditosAprobados(int creditos){
 creditosAprobados = creditos;
}

public float getCreditosAprobados(){
 return creditosAprobados;
}
```

Variables `Nombre` y `numDocumento` no tienen métodos `set` asociados

Clase inmutable: ningún atributo se puede modificar

Método toString

```
public String toString(){
                    return nombre + "("+numDocumento+")";
            }
```

Reemplaza (*overrides*) la definición por defecto en la clase `Object`

Usando la clase Estudiante

- Clase EstudianteGrado

  Crear nuevo objeto de la clase Estudiante:

```
String nombre = "Juan";
int numDoc = 40928123;
Estudiante juan = new Estudiante(nombre, numDoc);
```

  Especificar otros atributos del objeto:

```
juan.setCreditosAprobados(120);
juan.setAlDiaPagos(true);
```

  Usar otro constructor:

```
Estudiante otroJuan = new Estudiante("Juan␣F",
   10231234, 160, false);
```

Usando la clase Estudiante

Usar los métodos de la clase (y el método toString):

```
println("El␣estudiante␣"+ juan+"␣"+ (juan.isAlDiaPagos
    ()?"":"NO") +"␣se␣encuentra␣al␣día␣en␣pagos.\n");
```

Usar las constantes de la clase:

```
if(juan.getCreditosAprobados() >= Estudiante.
    CREDITOS_PARA_GRADO ){
                            println("El␣estudiante␣"+juan+"␣
    ha␣acumulado␣los␣créditos␣para␣graduarse");
                }else{
                            println("El␣estudiante␣"+juan+"␣
    ha␣acumulado␣"+juan.getCreditosAprobados()+"␣
    créditos.␣Aún␣no␣puede␣graduarse");
                }
```

Javadoc

Javadoc en la clase Estudiante

Encontramos descripciones de:

- La clase
- Constantes y variables públicas
- Constructores y sus parámetros
- Cada método, sus parámetros, el valor que retorna

Generando el Javadoc

Project → Generate Javadoc

Asegurarse que resultado va a carpeta `doc`

Tras generarlo, ir a carpeta `doc` y abrir archivo `index.html`

Extendiendo Clases

Extendiendo Clases

```
public class EstudianteMACC extends Estudiante{

 private String areaProf;
}
```

Extendiendo Clases - Constructor

```
        public EstudianteMACC(String nombre, int numDoc,
    String area){
                super(nombre, numDoc);
                areaProf = area;
        }
```

super: usamos el constructor de la súper-clase

Extendiendo Clases - Funcionalidad Adicional

```
public void setAreaProf(String area){
        areaProf = area;
}

public String getAreaProf(){
        return areaProf;
}
```