Objetos 2: Clases y métodos

Programación I - UNGS

Clases

- En Java, cada vez que creamos un programa, creamos una clase para él.

Clases

- En Java, cada vez que creamos un programa, creamos una clase para él.
- Dijimos que **los objetos** son una colección de datos y métodos relacionados.

Clases

- En Java, cada vez que creamos un programa, creamos una clase para él.
- Dijimos que **los objetos** son una colección de datos y métodos relacionados.
- **Las clases** son el "molde" a partir de la cual se crean los objetos, dado que especifican qué variables de instancia contienen y a qué métodos responden.

Clases

- En Java, cada vez que creamos un programa, creamos una clase para él.
- Dijimos que **los objetos** son una colección de datos y métodos relacionados.
- **Las clases** son el "molde" a partir de la cual se crean los objetos, dado que especifican qué variables de instancia contienen y a qué métodos responden.
- Los datos y métodos asociados a un tipo de objeto se definen en la clase, con lo cual podemos decir que la clase **representa** un tipo de objeto.

Clase Fecha

- Ejemplo: supongamos que queremos definir una clase para representar fechas.

Clase Fecha

- Ejemplo: supongamos que queremos definir una clase para representar fechas.
- Esencialmente, una fecha es un **día** de un **mes** de un **año**.

Clase Fecha

- Ejemplo: supongamos que queremos definir una clase para representar fechas.
- Esencialmente, una fecha es un **día** de un **mes** de un **año**.

```
1  public class Fecha {
2      int dia;
3      int mes;
4      int anio;
5  }
```

Clase Fecha

- Ejemplo: supongamos que queremos definir una clase para representar fechas.
- Esencialmente, una fecha es un **día** de un **mes** de un **año**.

```
1  public class Fecha {
2      int dia;
3      int mes;
4      int anio;
5  }
```

- ¿Cómo hacemos ahora para crear objetos de tipo Fecha?

Constructores

- Cuando se crea un objeto de una clase se llama a un método especial llamado **constructor**.

Constructores

- Cuando se crea un objeto de una clase se llama a un método especial llamado **constructor**.
- El constructor se encarga de inicializar las variables de instancia.

Constructores

- Cuando se crea un objeto de una clase se llama a un método especial llamado **constructor**.
- El constructor se encarga de inicializar las variables de instancia.
- Veamos un ejemplo de constructor para la clase Fecha.

Constructores

- Cuando se crea un objeto de una clase se llama a un método especial llamado **constructor**.
- El constructor se encarga de inicializar las variables de instancia.
- Veamos un ejemplo de constructor para la clase Fecha.

```
1  public Fecha() {
2      this.dia = 1;
3      this.mes = 1;
4      this.anio = 1970;
5  }
```

Ejemplo de Constructor

```
1 public Fecha() {
2     this.dia = 1;
3     this.mes = 1;
4     this.anio = 2000;
5 }
```

- Nótese que el constructor **no devuelve ningún valor**.

Ejemplo de Constructor

```
1 public Fecha() {
2     this.dia = 1;
3     this.mes = 1;
4     this.anio = 2000;
5 }
```

- Nótese que el constructor **no devuelve ningún valor**.
- La palabra **this** es una referencia al objeto que está siendo "construído".

Ejemplo de Constructor

```
1 public Fecha() {
2     this.dia = 1;
3     this.mes = 1;
4     this.anio = 2000;
5 }
```

- Nótese que el constructor **no devuelve ningún valor**.
- La palabra **this** es una referencia al objeto que está siendo "construído".
- Cuando escribimos **new** Fecha() en una expresión, se crea un objeto de tipo Fecha y se llama a este método sobre el objeto para inicializarlo.

Más de un construcror

- Podemos definir más de un constructor, siempre con parámetros distintos.

Más de un construcror

- Podemos definir más de un constructor, siempre con parámetros distintos.
- Por ejemplo:

```
1  public Fecha(int d, int m, int a) {
2      this.dia = d;
3      this.mes = m;
4      this.anio = a;
5  }
```

Más de un construcror

- Podemos definir más de un constructor, siempre con parámetros distintos.
- Por ejemplo:

```
1  public Fecha(int d, int m, int a) {
2      this.dia = d;
3      this.mes = m;
4      this.anio = a;
5  }
```

- Este constructor es un constructor trivial en el que pasamos un valor para cada variable de instancia.

Más de un construcror

- Podemos definir más de un constructor, siempre con parámetros distintos.
- Por ejemplo:

```
1  public Fecha(int d, int m, int a) {
2      this.dia = d;
3      this.mes = m;
4      this.anio = a;
5  }
```

- Este constructor es un constructor trivial en el que pasamos un valor para cada variable de instancia.
- Es muy común contar con este tipo de constructores.

Imprimiendo objetos

- ¿Qué pasa si ejecutamos el siguiente código?

```
1  public static void main(String[] args) {
2      Fecha f = new Fecha();
3      System.out.println(f);
4  }
```

Imprimiendo objetos

- ¿Qué pasa si ejecutamos el siguiente código?

```
1 public static void main(String[] args) {
2       Fecha f = new Fecha();
3       System.out.println(f);
4 }
```

- Obtenemos algo como Fecha@addbf1. ¿Por qué?

Imprimiendo objetos

- ¿Qué pasa si ejecutamos el siguiente código?

```
1 public static void main(String[] args) {
2       Fecha f = new Fecha();
3       System.out.println(f);
4 }
```

- Obtenemos algo como Fecha@addbf1. ¿Por qué?
- En principio, Java no sabe cómo mostrar objetos de tipo Fecha.

Imprimiendo objetos

- ¿Qué pasa si ejecutamos el siguiente código?

```
1  public static void main(String[] args) {
2      Fecha f = new Fecha();
3      System.out.println(f);
4  }
```

- Obtenemos algo como Fecha@addbf1. ¿Por qué?
- En principio, Java no sabe cómo mostrar objetos de tipo Fecha.
- Esta es la forma por defecto en que Java muestra los nuevos tipos.

Objetos autoimprimibles

- Nos gustaría agregar un método para que el objeto se imprima por pantalla en un formato entendible.

Objetos autoimprimibles

- Nos gustaría agregar un método para que el objeto se imprima por pantalla en un formato entendible.
- La idea sería que se invoque escribiendo: $\mathrm{f.imprimir}()$.

Objetos autoimprimibles

- Nos gustaría agregar un método para que el objeto se imprima por pantalla en un formato entendible.
- La idea sería que se invoque escribiendo: $\mathrm{f.imprimir()}$.

```
1  public class Fecha {
2      ...
3      public void imprimir() {
4          System.out.println(this.dia + "/" + this.mes + "/" + this.anio);
5      }
6  }
```

Objetos autoimprimibles

- Nos gustaría agregar un método para que el objeto se imprima por pantalla en un formato entendible.
- La idea sería que se invoque escribiendo: $\mathrm{f.imprimir()}$.

```
1  public class Fecha {
2      ...
3      public void imprimir() {
4          System.out.println(this.dia + "/" + this.mes + "/" + this.anio);
5      }
6  }
```

- Este método es un ejemplo de lo que se llama un **método de instancia**.

Métodos de **instancia**

- Nótese que este método **no lleva la palabra static**.

Métodos de **instancia**

- Nótese que este método **no lleva la palabra static**.
- Esto es lo que indica que un método es un método de instancia y el mismo debe ser invocado sobre un objeto de esta clase.

Métodos de **instancia**

- Nótese que este método **no lleva la palabra static**.
- Esto es lo que indica que un método es un método de instancia y el mismo debe ser invocado sobre un objeto de esta clase.
- El objeto sobre el que se invocan es el **parámetro implícito** (y es accesible mediante la palabra **this**).

Métodos de **instancia**

- Nótese que este método **no lleva la palabra static**.
- Esto es lo que indica que un método es un método de instancia y el mismo debe ser invocado sobre un objeto de esta clase.
- El objeto sobre el que se invocan es el **parámetro implícito** (y es accesible mediante la palabra **this**).
- Un ejemplo de un método de instancia para la clase Fecha podría ser un método que indique si la fecha es un día de verano o no.

Métodos de **instancia**

- Nótese que este método **no lleva la palabra static**.
- Esto es lo que indica que un método es un método de instancia y el mismo debe ser invocado sobre un objeto de esta clase.
- El objeto sobre el que se invocan es el **parámetro implícito** (y es accesible mediante la palabra **this**).
- Un ejemplo de un método de instancia para la clase Fecha podría ser un método que indique si la fecha es un día de verano o no.
- Dado un objeto f de la clase Fecha, podríamos preguntarle a f si representa un día de verano escribiendo $\mathrm{f.esVerano()}$.

Métodos de **instancia**

- El código podría ser algo así:

```
1  boolean esVerano() {
2    if (this.mes == 1 || this.mes == 2) {
3      return true;
4    else if (this.mes == 12 && this.dia >= 21)
5      return true;
6    else if (this.mes == 3 && this.dia < 21)
7      return true;
8    else
9      return false;
10 }
```

Métodos de **instancia**

- El código podría ser algo así:

```
1  boolean esVerano() {
2    if (this.mes == 1 || this.mes == 2) {
3      return true;
4    else if (this.mes == 12 && this.dia >= 21)
5      return true;
6    else if (this.mes == 3 && this.dia < 21)
7      return true;
8    else
9      return false;
10 }
```

- En este caso **this** representa al objeto f, es decir, a la instancia sobre la cual se llamó al método.

Métodos de **clase** (o estáticos)

- Hagamos ahora un método de la clase Fecha que nos diga si un año es o no bisiesto.

Métodos de **clase** (o estáticos)

- Hagamos ahora un método de la clase Fecha que nos diga si un año es o no bisiesto.

```
1  static boolean bisiesto(int anio) {
2    if (anio % 4 == 0 && anio % 100 != 0)
3      return true;
4    else if (anio % 400 == 0)
5      return true;
6    else
7      return false;
8  }
```

Métodos de **clase** (o estáticos)

- Hagamos ahora un método de la clase Fecha que nos diga si un año es o no bisiesto.

```
1  static boolean bisiesto(int anio) {
2    if (anio % 4 == 0 && anio % 100 != 0)
3      return true;
4    else if (anio % 400 == 0)
5      return true;
6    else
7      return false;
8  }
```

- En este caso el resultado del método no depende de ningún objeto, ya que un año es o no es bisiesto, sin importar de "a quién" se le pregunte.

Métodos de **clase** (o estáticos)

- El anterior es un **método de clase** ya que no depende de alguna **instancia** en particular.

Métodos de **clase** (o estáticos)

- El anterior es un **método de clase** ya que no depende de alguna **instancia** en particular.
- En un método de clase no existe el parámetro implícto **this** y el método no se invoca "sobre" un objeto.

Métodos de **clase** (o estáticos)

- El anterior es un **método de clase** ya que no depende de alguna **instancia** en particular.
- En un método de clase no existe el parámetro implícto **this** y el método no se invoca "sobre" un objeto.
- Por ejemplo, para listar los años bisiestos hasta el año 2000, podemos usar este método con la siguiente sintaxis:

```
1   for (int a = 1; a <= 2000; a++)
2     if (Fecha.bisiesto(a)) // <-- Llamado al método de clase!
3         System.out.println("El año " + a + " es bisiesto");
```

Métodos de **clase** (o estáticos)

- El anterior es un **método de clase** ya que no depende de alguna **instancia** en particular.
- En un método de clase no existe el parámetro implícto **this** y el método no se invoca "sobre" un objeto.
- Por ejemplo, para listar los años bisiestos hasta el año 2000, podemos usar este método con la siguiente sintaxis:

```
1   for (int a = 1; a <= 2000; a++)
2     if (Fecha.bisiesto(a)) // <-- Llamado al método de clase!
3         System.out.println("El año " + a + " es bisiesto");
```

- Un método de clase puede escribirse también como un método de instancia, pero para llamar al método necesitaríamos contar con una instancia desde la cual llamarlo (aunque el método no utilice la instancia implícita).

En el libro...

Lo que vimos en esta clase lo pueden encontrar en el **Capítulo 9** del libro.