Objetos 3: Invariante de representación y Visibilidad

Programación I - UNGS

Los objetos y sus variables de instancia

- Una instancia representa un concepto (válido) de una clase (Rectangle, Fecha, etc.).

Los objetos y sus variables de instancia

- Una instancia representa un concepto (válido) de una clase (Rectangle, Fecha, etc.).
- Las variables de instancia de un objeto representan su estado actual.

Los objetos y sus variables de instancia

- Una instancia representa un concepto (válido) de una clase (Rectangle, Fecha, etc.).
- Las variables de instancia de un objeto representan su estado actual.
- Cualquier concepto válido de una clase (Rectangle, Fecha, etc.) puede ser representado por un conjunto de valores para sus variables de instancia.

Los objetos y sus variables de instancia

- Una instancia representa un concepto (válido) de una clase (Rectangle, Fecha, etc.).
- Las variables de instancia de un objeto representan su estado actual.
- Cualquier concepto válido de una clase (Rectangle, Fecha, etc.) puede ser representado por un conjunto de valores para sus variables de instancia.
- ¡Pero no ocurre lo mismo al revés!

Ejemplo

- Fabricamos nuestra clase Fecha y alguien (quizás nosotros mismos) la usamos en el siguiente código:

```
1  Fecha f = new Fecha();
2
3  // hago cosas con f...
4
5  // avanzo un día
6  f.dia++;
```

Ejemplo

- Fabricamos nuestra clase Fecha y alguien (quizás nosotros mismos) la usamos en el siguiente código:

```
1  Fecha f = new Fecha();
2
3  // hago cosas con f...
4
5  // avanzo un día
6  f.dia++;
```

- ¿Está bien este código?

Ejemplo

- Fabricamos nuestra clase Fecha y alguien (quizás nosotros mismos) la usamos en el siguiente código:

```
1  Fecha f = new Fecha();
2
3  // hago cosas con f...
4
5  // avanzo un día
6  f.dia++;
```

- ¿Está bien este código?
- ¿Qué pasa si f era el 31/1/2000?

Ejemplo

- Fabricamos nuestra clase Fecha y alguien (quizás nosotros mismos) la usamos en el siguiente código:

```
1 Fecha f = new Fecha();
2
3 // hago cosas con f...
4
5 // avanzo un día
6 f.dia++;
```

- ¿Está bien este código?
- ¿Qué pasa si f era el 31/1/2000?
- El objeto quedaría de alguna forma... "roto".

Invariante de representación

Cuando un objeto está "roto", decimos que no se cumple su Invariante de representación...

> *Conjunto de propiedades que deben cumplir los "datos internos" de un objeto para asegurar que dicho objeto representa un "concepto" válido.*

Invariante de representación

Cuando un objeto está "roto", decimos que no se cumple su Invariante de representación...

> *Conjunto de propiedades que deben cumplir los "datos internos" de un objeto para asegurar que dicho objeto representa un "concepto" válido.*

- Un objeto "roto" eventualmente trae problemas graves.

Invariante de representación

Cuando un objeto está "roto", decimos que no se cumple su Invariante de representación...

> *Conjunto de propiedades que deben cumplir los "datos internos" de un objeto para asegurar que dicho objeto representa un "concepto" válido.*

- Un objeto "roto" eventualmente trae problemas graves.
- ¡No debemos permitir que existan objetos "rotos" en nuestros programas!

Invariante de representación

Cuando un objeto está "roto", decimos que no se cumple su Invariante de representación...

> *Conjunto de propiedades que deben cumplir los "datos internos" de un objeto para asegurar que dicho objeto representa un "concepto" válido.*

- Un objeto "roto" eventualmente trae problemas graves.
- ¡No debemos permitir que existan objetos "rotos" en nuestros programas!
- ¿Cómo hacemos para asegurar que eso no va a pasar con nuestros objetos?

Volvamos al ejemplo

```
1  Fecha f = new Fecha();
2
3  // hago cosas con f...
4
5  // avanzo un día
6  f.dia++;
```

Volvamos al ejemplo

```
1  Fecha f = new Fecha();
2
3  // hago cosas con f...
4
5  // avanzo un día
6  f.dia++;
```

- ¿Está bien este código?

Volvamos al ejemplo

```
1  Fecha f = new Fecha();
2
3  // hago cosas con f...
4
5  // avanzo un día
6  f.dia++;
```

- ¿Está bien este código?
- ¿Cómo lo soluciono?

Volvamos al ejemplo

```
1 Fecha f = new Fecha();
2
3 // hago cosas con f...
4
5 // avanzo un día
6 f.dia++;
```

- ¿Está bien este código?
- ¿Cómo lo soluciono?
- ¿Cómo evito que esto vuelva a pasar?

Visibilidad

- Cuando declaramos una variable de instancia, definimos su visibilidad:

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

Visibilidad

- Cuando declaramos una variable de instancia, definimos su visibilidad:

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

- Cuando una variable es privada, sólo es accesible desde dentro de la clase.

Visibilidad

- Cuando declaramos una variable de instancia, definimos su visibilidad:

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

- Cuando una variable es privada, sólo es accesible desde dentro de la clase.
- ¿Qué pasa con el código del ejemplo?

Visibilidad

- Cuando declaramos una variable de instancia, definimos su visibilidad:

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

- Cuando una variable es privada, sólo es accesible desde dentro de la clase.
- ¿Qué pasa con el código del ejemplo?
- ¿Como hacemos para ver el valor de una variable privada desde afuera de la clase?

Visibilidad

- La “publicamos” (o *exportamos*) mediante una función:

Visibilidad

- La "publicamos" (o *exportamos*) mediante una función:

```
1  public class Fecha {
2      private int dia;
3      private int mes;
4      private int anio;
5
6      public int getDia() {
7          return this.dia;
8      }
9  }
```

Visibilidad

- La "publicamos" (o *exportamos*) mediante una función:

```
1  public class Fecha {
2      private int dia;
3      private int mes;
4      private int anio;
5
6      public int getDia() {
7          return this.dia;
8      }
9  }
```

```
1  // Código fuera de la clase Fecha
2
3  Fecha f = new Fecha();
4
5  // hago cosas con f...
6
7  // consulto el día de f
8  int d = f.getDia();
```

Visibilidad

- La "publicamos" (o *exportamos*) mediante una función:

```
1  public class Fecha {
2      private int dia;
3      private int mes;
4      private int anio;
5
6      public int getDia() {
7          return this.dia;
8      }
9  }
```

```
1  // Código fuera de la clase Fecha
2
3  Fecha f = new Fecha();
4
5  // hago cosas con f...
6
7  // consulto el día de f
8  int d = f.getDia();
```

A estos métodos se los suele llamar *getters*...

Visibilidad

- Ahora podemos “ver” una variable privada desde afuera de la clase.

Visibilidad

- Ahora podemos “ver” una variable privada desde afuera de la clase.
- ¿Y si la quiero editar?

Visibilidad

- Ahora podemos "ver" una variable privada desde afuera de la clase.
- ¿Y si la quiero editar? Hacemos algo similar...

```
1  public class Fecha {
2      private int dia;
3      private int mes;
4      private int anio;
5
6      public void setDia(int d) {
7          this.dia = d;
8      }
9  }
```

Visibilidad

- Ahora podemos "ver" una variable privada desde afuera de la clase.
- ¿Y si la quiero editar? Hacemos algo similar...

```
1  public class Fecha {
2     private int dia;
3     private int mes;
4     private int anio;
5
6     public void setDia(int d) {
7        this.dia = d;
8     }
9  }
```

```
1        // Código fuera de la clase Fecha
2
3        Fecha f = new Fecha();
4
5        // hago cosas con f...
6
7        // avanzo un día
8        f.setDia(f.getDia()+1);
```

Visibilidad

- Ahora podemos "ver" una variable privada desde afuera de la clase.
- ¿Y si la quiero editar? Hacemos algo similar...

```
1  public class Fecha {
2     private int dia;
3     private int mes;
4     private int anio;
5
6     public void setDia(int d) {
7        this.dia = d;
8     }
9  }
```

```
1        // Código fuera de la clase Fecha
2
3        Fecha f = new Fecha();
4
5        // hago cosas con f...
6
7        // avanzo un día
8        f.setDia(f.getDia()+1);
```

A estos métodos se los suele llamar *setters*...

Visibilidad

- Pero entonces... ¿qué cambió?

Visibilidad

- Pero entonces... ¿qué cambió?
- Ahora podemos controlar los valores:

```
1  public class Fecha {
2    // ...
3
4    public void setDia(int d) {
5      int diasMes = diasDelMes(this.mes, this.anio);
6      if (d < 1 || d > diasMes) {
7        throw new RuntimeException("El día es inválido para " + this);
8      }
9
10     this.dia = d;
11   }
12 }
```

Visibilidad

La visibilidad de un miembro (variable o método) de una clase indica desde qué clases es accesible. Las opciones posibles son:

- **private** (privada): visibles sólo dentro de la misma clase.
- **public** (pública): visibles desde cualquier parte del proyecto.
- **package** (paquete): visibles sólo desde clases que estén dentro del mismo paquete que la clase en cuestión. Esta visibilidad es la que se asume si no se pone nada.
- **protected** (protegida): visibles desde la misma clase y cualquiera de sus "subclases" (este tema se ve en la materia Programación II).

Visibilidad

La visibilidad de un miembro (variable o método) de una clase indica desde qué clases es accesible. Las opciones posibles son:

- **private** (privada): visibles sólo dentro de la misma clase.
- **public** (pública): visibles desde cualquier parte del proyecto.
- **package** (paquete): visibles sólo desde clases que estén dentro del mismo paquete que la clase en cuestión. Esta visibilidad es la que se asume si no se pone nada.
- **protected** (protegida): visibles desde la misma clase y cualquiera de sus "subclases" (este tema se ve en la materia Programación II).

Idealmente, nuestras clases deberían asegurar que los objetos de la misma no se pueden "romper". Es decir, que la clase es robusta.

Visibilidad

La visibilidad de un miembro (variable o método) de una clase indica desde qué clases es accesible. Las opciones posibles son:

- **private** (privada): visibles sólo dentro de la misma clase.
- **public** (pública): visibles desde cualquier parte del proyecto.
- **package** (paquete): visibles sólo desde clases que estén dentro del mismo paquete que la clase en cuestión. Esta visibilidad es la que se asume si no se pone nada.
- **protected** (protegida): visibles desde la misma clase y cualquiera de sus "subclases" (este tema se ve en la materia Programación II).

Idealmente, nuestras clases deberían asegurar que los objetos de la misma no se pueden "romper". Es decir, que la clase es robusta.

La visibilidad puede usarse tanto para variables como para métodos, ya sean de instancia o de clase.

En resumen

- Permitir que se modifiquen las variables de instancia de un objeto es peligroso.

En resumen

- Permitir que se modifiquen las variables de instancia de un objeto es peligroso.
- Puede llevar a que estos objetos no cumplan con su invariante de representación.

En resumen

- Permitir que se modifiquen las variables de instancia de un objeto es peligroso.
- Puede llevar a que estos objetos no cumplan con su invariante de representación.
- Una forma de evitar eso es hacer clases robustas "ocultando" datos internos (¡o métodos frágiles!).

En resumen

- Permitir que se modifiquen las variables de instancia de un objeto es peligroso.
- Puede llevar a que estos objetos no cumplan con su invariante de representación.
- Una forma de evitar eso es hacer clases robustas "ocultando" datos internos (¡o métodos frágiles!).
- Es muy usual que las variables de instancia sean siempre privadas y se manejen mediante *getters* y *setters* (aun cuando estos no hagan nada... ¿por qué?).