Sesión 11: Tipos Abstractos de Datos

Programación 2

4. Tipos Abstractos de Datos

Ángel Herranz

2019-2020

Universidad Politécnica de Madrid

Notas para el siguiente semestre

- La clase es un poco extraña, quizás un poco corta.
- No queda suficientemente claro que lo mas importante en esta sesión es el concepto de TAD.
- Quizás sea necesario mencionar que asociado al TAD están sus tests (UNIT TESTING).
- Y que asociado al TAD está su documentación (inseparables), supernecesaria para desarrollar el código cliente.

En capítulos anteriores

- Tema 1: Clases y Objetos
- Tema 2: Colecciones acotadas de Objetos

En capítulos anteriores

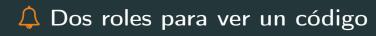
- Tema 1: Clases y Objetos
- Tema 2: Colecciones acotadas de Objetos
 - Todo con arrays nativos: T[]
 - Nulles
 - Índices
 - Pilas acotadas

Tema 2: Colecciones acotadas de Objetos

- Tema 3: Programación modular
- Tema 4: Tipos Abstractos de Datos

- Tema 3: Programación modular
- Tema 4: Tipos Abstractos de Datos

SIMs



Cuando lo implementas

У

Cuando lo usas

SIMs

- La idea es hacer un simulador social
- Las personas (SIMs) tienen un nombre (añade otras características, como género)
- Cada SIM puede dormir, comer, trabajar, jugar, hablar, . . .
- Cada SIM puede hacerse amigo de otro SIM
- El simulador hace pasar el tiempo en pulsos de una hora

Cuando usas un código...

Así ve la clase SIM el programador del simulador

Cuando usas un código...

Así ve la clase SIM el programador del simulador

```
public class SIM {
  public enum Actividad { DORMIR, COMER, ... };
  public SIM(String nombre) ...
  public void simular(int tiempoHoras) ...
  public void hacerAmigo(SIM amigo) ...
  public String nombre() ...
  public Actividad quease() ...
  public int estadistica(Actividad a) ...
  public SIM amigo() ...
```

...sólo ves su API

TAD

Tipo Abstracto de Datos

...sólo ves su API

TAD

Tipo Abstracto de Datos Abstract Data Type (ADT)

... sólo ves su API

TAD

Tipo Abstracto de Datos Abstract Data Type (ADT)

- No quieres saber cómo está hecho
- Sólo quieres saber qué hace: semántica

... sólo ves su API

TAD

Tipo Abstracto de Datos Abstract Data Type (ADT)

- No quieres saber cómo está hecho
- Sólo quieres saber qué hace: semántica
- ¿Entiendes la semántica de cada método?

... sólo ves su API

TAD

Tipo Abstracto de Datos Abstract Data Type (ADT)

- No quieres saber cómo está hecho
- Sólo quieres saber qué hace: semántica
- ¿Entiendes la semántica de cada método?
- ¿Y la semántica del todo?

Cuando implementas un código...

Así ve la clase SIM el implementador de SIM. java

```
import java.util.Random;
public class SIM {
   public enum Actividad {
       DORMIR, COMER, ESTUDIAR, JUGAR
   };
   private String nombre;
   private Actividad haciendo;
   private int horas;
   private SIM amigo;
```

private int[] dedicado:

```
public SIM(String nombre) {
    this.nombre = nombre;
    haciendo = Actividad.DORMIR;
    horas = 0;
    amigo = null;
    int n = Actividad.values().length;
    dedicado = new int[n];
```

```
public void simular(int tiempoHoras) {
 Actividad siquiente = null;
 horas = horas + tiempoHoras:
 switch (haciendo) {
 case DORMIR:
   if (horas < 8)
      siquiente = Actividad.DORMIR:
    else
      siquiente = Actividad.COMER;
    break:
 case ESTUDIAR:
    if (horas < 8)
      siquiente = Actividad.ESTUDIAR;
    break:
 default:
    break:
```

Cuando implementas un código...

Así ve la clase SIM el implementador de SIM. java

```
import java.util.Random;
public class SIM {
   public enum Actividad {
       DORMIR, COMER, ESTUDIAR, JUGAR
   };
   private String nombre;
   private Actividad haciendo;
   private int horas;
   private SIM amigo;
```

private int[] dedicado:

```
public SIM(String nombre) {
    this.nombre = nombre;
    haciendo = Actividad.DORMIR;
    horas = 0;
    amigo = null;
    int n = Actividad.values().length;
    dedicado = new int[n];
}
```

```
public void simular(int tiempoHoras) {
 Actividad siquiente = null;
 horas = horas + tiempoHoras:
 switch (haciendo) {
 case DORMIR:
   if (horas < 8)
      siquiente = Actividad.DORMIR:
    else
      siquiente = Actividad.COMER;
    break:
 case ESTUDIAR:
    if (horas < 8)
      siquiente = Actividad.ESTUDIAR:
    break:
 default:
    break:
```

... sólo te preocupa el cómo

Modelización Estructuras de Datos (en forma de *atributos privados*)

...sólo te preocupa el <mark>cómo</mark>

Modelización
Estructuras de Datos
(en forma de *atributos privados*)

- Estás al servicio de quienes lo usan
- para que no necesiten mirar el cómo

... sólo te preocupa el cómo

Modelización Estructuras de Datos (en forma de *atributos privados*)

- Estás al servicio de quienes lo usan
- para que no necesiten mirar el cómo
- ¿Entiendes la modelización?
- ¿Para qué está cada atributo?

¿Y el qué?

¿Cómo explicamos la semántica?

¹Quizás necesites la opción -charset utf8 y para más tarde -author

¿Y el qué?

¿Cómo explicamos la semántica?

- Por el momento, sólo con comentarios
- Pero vamos a *javadocumentar*¹

```
C:\ Sesion11> javadoc -d doc_sim SIM.java
Loading source file SIM.java...
```

. . .

Generating doc_sim/help-doc.html...

¹Quizás necesites la opción -charset utf8 y para más tarde -author

Javadoc i

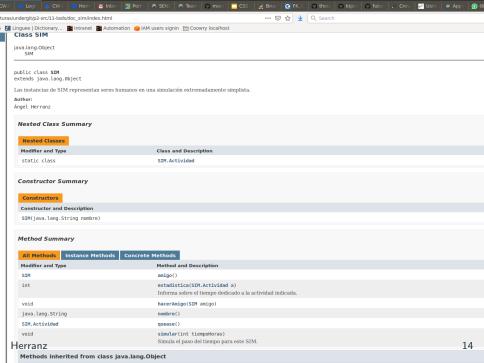
```
/**
 * Las instancias de SIM representan seres humanos en una simulación
 * extremadamente simplista.
 *
 * @author Ángel Herranz
 */
public class SIM {
    ...
```

Javadoc ii

```
/**
 * Simula el paso del tiempo para este SIM. El SIM, con el paso del
 * tiempo, cambia de actividad y acumula la información de la
 * actividad anterior.
 *
 * @param tiempoHoras tiempo simulado (en horas)
 */
public void simular(int tiempoHoras) {
 ...
}
```

Javadoc iii

```
/**
 * Informa sobre el tiempo dedicado a la actividad indicada.
 * @param a actividad sobre la que se quiere conocer el tiempo
 * dedicado
 * @return número de horas dedicadas a la actividad indicada
*/
public int estadistica(Actividad a) {
```



Entonces ¿qué es un TAD?

Un TAD (tipo abstracto de datos) lo definen

- Un nombre, el nombre del tipo.
- Unas operaciones.
- La semántica de las operaciones.

Entonces ¿qué es un TAD?

Un TAD (tipo abstracto de datos) lo definen

- Un nombre, el nombre del tipo.
- Unas operaciones.
- La semántica de las operaciones.
- El mismo TAD puede tener múltiples implementaciones concretas: estructuras de datos.

Desarrollo dirigido por los tests

Desarrollo dirigido por los tests ③

Desarrollo dirigido por los tests **?**Antes de implementar nada,
implementamos los tests

Desarrollo dirigido por los tests **?**Antes de implementar nada,
implementamos los tests

- Lo aplicamos a este ejemplo
 - 1. Escribimos el código de SIM. java "vacío", sólo para que compile
 - Escribimos un programa principal TestSIM.java que compruebe propiedades de acuerdo a la semántica

SIM. java "vacío"

```
public class SIM {
  public enum Actividad {
    DORMIR, COMER, ESTUDIAR, JUGAR
  };
  public SIM(String nombre) {
  public void simular(int tiempoHoras) {
  public String nombre() {
    return "Ángel";
```

```
public void hacerAmigo(SIM amigo) {
public SIM amigo() {
  return null;
public Actividad guease() {
  return Actividad.DORMIR:
public int estadistica(Actividad a) {
  return 0:
```

```
public class TestSIM {
  public static void main(String[] ags) {
    SIM lucia = new SIM("Lucía");
    assert lucia.quease() == SIM.Actividad.DORMIR;
    lucia.simular(1);
    assert lucia.quease() == SIM.Actividad.DORMIR;
    assert "Lucía".equals(lucia.nombre());
    lucia.simular(8);
    assert lucia.quease() == SIM.Actividad.COMER;
```

```
public class TestSIM {
  public static void main(String[] ags) {
    SIM lucia = new SIM("Lucía");
    assert lucia.quease() == SIM.Actividad.DORMIR;
    lucia.simular(1);
    assert lucia.quease() == SIM.Actividad.DORMIR;
    assert "Lucía".equals(lucia.nombre());
    lucia.simular(8):
    assert lucia.quease() == SIM.Actividad.COMER;
```

Compilar y ejecutar

```
public class TestSIM {
  public static void main(String[] ags) {
    SIM lucia = new SIM("Lucía");
    assert lucia.quease() == SIM.Actividad.DORMIR;
    lucia.simular(1);
    assert lucia.quease() == SIM.Actividad.DORMIR;
    assert "Lucía".equals(lucia.nombre());
    lucia.simular(8):
    assert lucia.quease() == SIM.Actividad.COMER;
```

Compilar y ejecutar con java -ea TestSIM

```
public class TestSIM {
  public static void main(String[] ags) {
    SIM lucia = new SIM("Lucía");
    assert lucia.quease() == SIM.Actividad.DORMIR;
    lucia.simular(1);
    assert lucia.quease() == SIM.Actividad.DORMIR;
    assert "Lucía".equals(lucia.nombre());
    lucia.simular(8):
    assert lucia.quease() == SIM.Actividad.COMER;
```

- Compilar y ejecutar con java -ea TestSIM
- Escribir más tests

```
public class TestSIM {
  public static void main(String[] ags) {
    SIM lucia = new SIM("Lucía");
    assert lucia.quease() == SIM.Actividad.DORMIR;
    lucia.simular(1);
    assert lucia.quease() == SIM.Actividad.DORMIR;
    assert "Lucía".equals(lucia.nombre());
    lucia.simular(8):
    assert lucia.quease() == SIM.Actividad.COMER;
```

- Compilar y ejecutar con java -ea TestSIM
- Escribir más tests antes de programar SIM. java

Tecnología para hacer TDD: assert

- Ya veremos tecnología más elaborada para hacer testing (ej. JUnit)
- De momento usamos el **assert** de Java:²

```
assert Condición [: Mensaje];
```

assert lucia.guease() == SIM.Actividad.DORMIR

Ejemplos de uso:

```
: "lucia debería estar durmiendo";

assert "Lucía".equals(lucia.nombre())
: "Nombre de lucia incorrecto: " + lucia.nombre();
```

²Los corchetes indican que el mensaje es opcional