

Introducción al diseño de tipos

Creación “Exprés” de una clase.

Restricciones

Fundamentos de Programación

Departamento de Lenguajes y Sistemas Informáticos



2024/25

El tipo Object (*Ejercicio: Persona*)

Corregimos que quedó pendiente.

Para el tipo *Persona*. Se trata ahora de implementar y probar los métodos *toString()*, *equals()* y *hashCode()*

Decidimos que:

- La representación textual de una persona es su dni, apellidos y nombre separados por punto y coma y un blanco y entre paréntesis:
(12345678A; García Gómez; Ana)
- Dos Personas son iguales si tienen el *mismo dni, nombre y apellidos*



2024/25

Criterio de orden natural (Persona)

Corregimos que quedó pendiente.

Orden natural de Persona

Diremos que el criterio de orden natural del tipo Persona es por el dni



2024/25

El tipo Object (*Ejercicio: Animal*)

Corregimos que quedó pendiente.

Para el tipo *Animal*. Se trata ahora de implementar y probar los métodos *toString()*, *equals()* y *hashCode()*

Decidimos que:

- La representación textual de un animal es su nombre seguido de la familia entre corchetes:

Buitre[AVE]

- Dos animales son iguales si tienen el *mismo nombre y familia*



2024/25

Criterio de orden natural (Animal)

Corregimos que quedó pendiente.

Orden natural de Animal

Diremos que el criterio de orden natural del tipo Animal es por el pesoMedio y a igualdad de pesoMedio por edadMedia y a igualdad de edadMedia por la familia



2024/25

Creación EXPRÉS de una clase

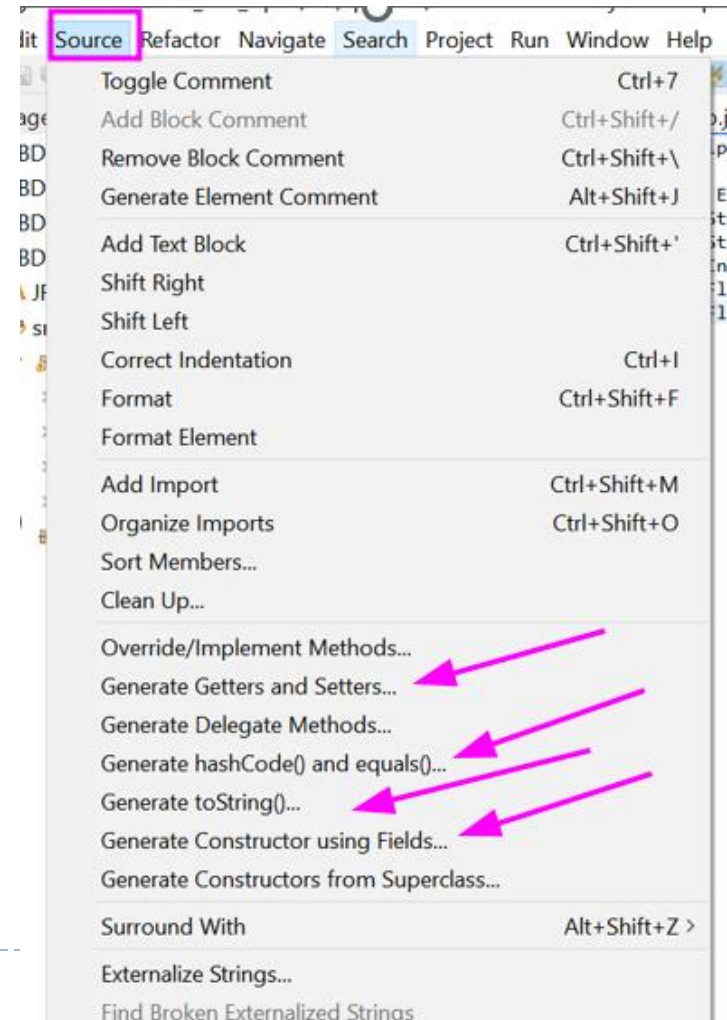
Eclipse nos proporciona una serie de utilidades que no van a permitir construir una clase de una forma rápida y cómoda a partir de los atributos de la clase.

Paso 1:

Crear la clase e implementar sólo los atributos (*propiedades básicas*)

Paso 2:

En la opción de menú “source” elegir las entradas apropiadas





2024/25

Creación EXPRÉS de una clase

Ejercicio de implementación “expres” de un tipo.

Proyecto: *02_tipos* Paquete: *fp.tipos* Clase: *Electrodoméstico*

Propiedades:

- Número de Serie. String. Consultable.
- Denominación: String. Consultable.
- Meses de garantía: Integer. Consultable
- Importe: Float. Consultable y modificable.
- Iva: Float. Consultable y modificable.
- Importe Total: Float. Consultable.

Métodos Constructores:

- Constructor_1: Debe tener un parámetro por cada propiedad básica.
- Constructor_2: Únicamente tendrá como parámetros el número de serie y la denominación. El resto de las propiedades básicas se inicializan a cero.

Métodos consultores y modificadores:

- Según se indica en la propiedad (*consultable y/o modificable*)



2024/25

Creación EXPRÉS de una clase

Ejercicio de implementación “expres” de un tipo.

Proyecto: *02_tipos* Paquete: *fp.tipos* Clase: *Electrodoméstico*

Representación textual:

- Todas las propiedades.

Criterio de Igualdad:

- Por el número de serie y el importe

Propiedades derivadas:

- Según indique el enunciado: En este caso el *Importe Total* que se calcula como el importe más el importe * el % de IVA .

Criterio de ordenación

(este método no se puede realizar de forma “expres”. Hay que programarlos

- Por el número de serie y a igualdad de número de serie desempata por el importe



2024/25

Creación EXPRÉS de una clase

Probando el tipo *Electrodoméstico*

Paquete: *fp.tipos.test* Clase: *TestElectrodoméstico*

Método *main*:

- Crear un objeto *e1* de tipo *Electrodoméstico* con el siguiente estado:
"1111","Batidora",24,50.0,21.0
- Crear un objeto *e2* de tipo *Electrodoméstico* con el siguiente estado:
"2222","Centro de Planchado",36,315.95,10.0
- Visualizar los dos objetos creados (*e1* y *e2*).
- Modificar el "iva" del segundo (*e2*) del 10% al 21%
- Visualizar la propiedad "Importe total" del segundo (*e2*)



2024/25

Implementación de restricciones

En la mayoría de los tipos que se implementan existen determinadas propiedades que no deberían tomar ciertos valores para que los objetos tengan sentido en el mundo real.

Por ejemplo:

- Una circunferencia no debe tener radio cero o negativo.
- Una persona no debe tener los apellidos y el nombre vacíos. O no puede nadie haber nacido en una fecha posterior al día de hoy.
- Un animal no debe tener pesoMedio o edadMedia negativo, o no puede tener el nombre en blanco.

Al hecho de controlar que los objetos no puedan tomar esos valores incorrectos lo denominamos *implementación de restricciones*.



2024/25

Implementación de restricciones

Implementación de restricciones

- Para controlar las restricciones, Java dispone de un mecanismo que se denomina lanzamientos de objetos excepciones (*throw*) que hace que el programa *detenga* su *ejecución*, informando al usuario por la consola de la restricción incumplida.

Sintaxis:

```
if (se incumple la restricción) {  
    throw new nombre_de_excepción ("Texto informativo");  
}
```



2024/25

Implementación de restricciones

Tipos Excepción más habituales:

En Java, *podemos implementar nuestros propios tipos de excepciones*, pero usaremos algunos ya implementados en JRE, por ejemplo:

- *IllegalArgumentException* (*normalmente usaremos esta*)
- *NullPointerException* (*y esta, también*)
- *ArithmeticException*
- *IndexOutOfBoundsException*
- *NoSuchElementException*
- ...



2024/25

Implementación de restricciones

¿Dónde se deben implementar las restricciones en una clase?

En aquellos métodos que asignen o modifiquen una propiedad con restricciones.

- Normalmente en los *constructores* y métodos *modificadores* que manejen la propiedad en la que se quiere controlar que no almacenen datos incorrectos.
- *NUNCA* en los métodos “*getters*” o en otros que devuelvan *propiedades derivadas*.

Se trata de que un objeto no almacene un *ESTADO* con un valor incorrecto en alguna/s de sus propiedades. *Por lo que, en definitiva, podemos decir que las restricciones se comprueban en los métodos que asignan valores a los atributos.*



2024/25

Implementación de restricciones

Ejemplo:

Restricciones del tipo Circunferencia:

- El radio debe ser positivo.

En su caso, lanzar la excepción *IllegalArgumentException* con el mensaje “*El radio debe ser positivo*”)

(recordar que *Circunferencia* tiene dos implementaciones y hay que hacer lo mismo en las dos).

En los *constructores* y en los métodos *setRadio(...)*

```
if (radio<=0) {  
    throw new IllegalArgumentException  
        ("El radio debe ser positivo");  
}
```

Ea, a modificar *CircunferencialImpl1*, *CircunferencialImpl2*, ...



2024/25

Implementación de restricciones

Ejemplo:

Probando las restricciones de Circunferencia:

Hacer un *testCircunferencia* en *geometría.test* en el que:

1. Se cree una circunferencia *c1* con radio 0, por ejemplo, con la implementación 2 y otra *c2* con radio 3.6 con la implementación 1. *En el fondo da igual las implementaciones que usemos*
2. Visualizar las dos circunferencias.
3. Se modifica el radio de *c1* a -20.
4. Visualizar otra vez las dos circunferencias.

Ejecutar el test y ver en la consola como salta la excepción.

Comentar la construcción de la segunda circunferencia y *volver a ejecutar el test*



2024/25

Implementación de restricciones

¿Dónde se deben implementar las restricciones en los record?,

En caso de los **record** en el que sólo se asignan valores a los atributos en los constructores - hay que recordar que no hay “setters” ya que los objetos de un tipo record son inmutables-, se hace mediante un “**bloque etiquetado**”.

Ejemplo en Persona:

```
public record Persona (String dni, String nombre,  
    String apellidos, LocalDate fechaNacimiento) {  
    ...  
    public Persona{  
        if (fechaNacimiento.isAfter(LocalDate.now())) {  
            throw new IllegalArgumentException("Error en la  
                fecha de nacimiento");  
        }  
    }  
    ...  
}
```




2024/25

Clases de utilidad

- Existen otras clases que no construyen ningún objeto y se llaman *clases de utilidad*. Estas clases no suelen tener atributos, sino una serie de métodos *static* que, para invocarlos, al carecer de objetos con los que invocarlos, se utiliza el *nombre de la clase*.

Por ejemplo:

- Math* (contiene métodos y constantes para trabajar con operaciones matemáticas).
 - Math.sqrt(...)* → forma de invocar al método que calcula la $\sqrt{\quad}$
- Checkers* (contiene métodos para comprobar las restricciones de las propiedades de los objetos)
 - Checkers.checkNotNull(...)* → forma de invocar al método que comprueba que los parámetros no sean nulos



2024/25

Implementación de restricciones

UNA GENTILEZA DEL DEPARTAMENTO

Clase de *utilidad Checkers* vs *throw new*

Hemos visto que para lanzar excepciones usamos una estructura

```
if (se incumple La restricción) {  
    throw new nombre_excepción ("Texto informativo");  
}
```

Si embargo, en la práctica, en nuestros proyectos usaremos la clase de utilidad *Checkers* con los métodos *checkNotNull* o *check*

Esta clase no forma parte de la API de Java, sino que la regalamos a los alumnos y a las alumnas de FP.



2024/25

Implementación de restricciones

Clase utilidad *Checkers*

Sintaxis

- *Checkers.checkNotNull* (parámetro1, parámetro2, parámetros3...);
- *Checkers.check* ("Mensaje informativo", restricción a cumplir);

Ejemplo de lanzamiento de una excepción si el centro o el radio de una circunferencia fuese nulo:

```
Checkers.checkNotNull(centro, radio);
```

Ejemplo de lanzamiento de una excepción si el radio es negativo.

```
Checkers.check("El radio debe ser positivo", radio > 0);
```

En lugar de

```
if (radio <= 0) {  
    throw new IllegalArgumentException  
        ("El radio debe ser positivo");  
}
```



2024/25

Implementación de restricciones

Captura de excepciones

Ya se ha explicado que el lanzamiento de excepciones *interrumpe* el *flujo* del programa y el *programa se aborta*, deteniendo su ejecución.

*En el caso de circunferencia hemos tenido que comentar la construcción de **c1** y el manejo de c1 para llegar a la construcción de **c2***

Java también dispone de otro mecanismo para que gestionemos las excepciones sin que el programa se detenga. Para esto último se usan los denominados bloques *try-catch* que permiten *capturar* las excepciones lanzadas. (*esto lo usaremos en los test*).

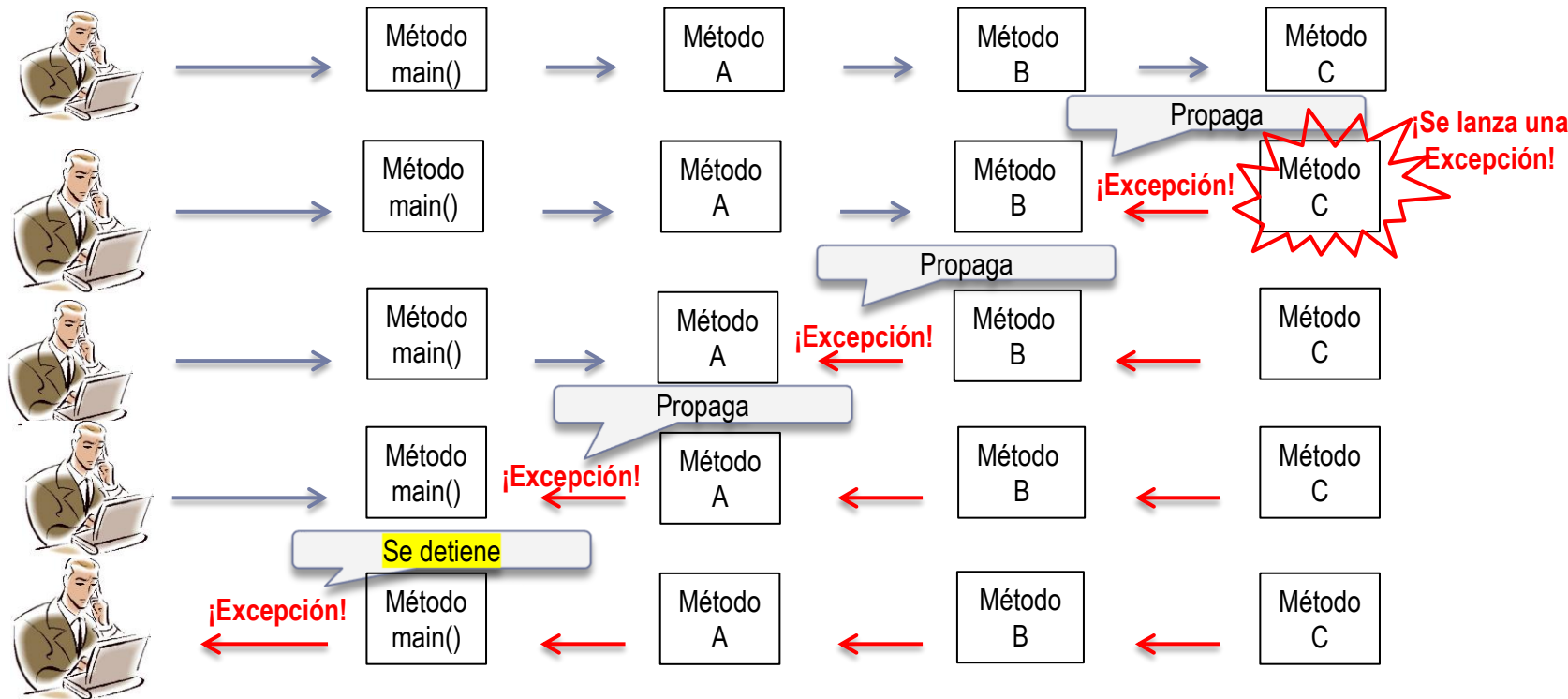


2024/25

Propagación de Excepciones

Propagar la excepción:

El método aborta su ejecución y el programa se detiene. Por ejemplo:



```
java.lang.IllegalArgumentException: El radio debe ser positivo
at geometría.CircunferencialImpl2.<init>(CircunferencialImpl2.java:9)
at geometría.test.TestCircunferencia6.main(TestCircunferencia6.java:8)
```



2024/25

Captura de Excepciones try-catch

La manera de evitar que se propague la excepción es mediante su captura y gestión:

Capturar la excepción: .

Es necesario incluir en el método que se quiere estudiar la excepción incluir un bloque *try-catch*. Sólo es obligatorio un try con, al menos, un bloque catch

```
try{  
    // Sentencias que incluyen llamadas a métodos  
    // que pueden generar excepciones.  
}catch(ExcepcionTipo1 e){  
    // Sentencias de tratamiento de la excepción tipo 1.  
}catch(ExcepcionTipo2 e){  
    // Sentencias de tratamiento de la excepción tipo 2.  
}finally{  
    // Sentencias que se ejecutan siempre al final.  
}
```



2024/25

Captura de Excepciones try-catch

Ejemplo:

Captura en el tipo Circunferencia:

Normalmente en nuestra asignatura las capturaremos en los test con el siguiente esquema:

```
public class TestCircunferencia_n {  
    public static void main(String[] args) {  
        testConstructor();  
        testSetRadio();  
    }  
    private static void testConstructor() {  
        Punto centro=new Punto(1.0,-1.0);  
        System.out.println("\nTEST del Constructor correcto");  
        try{  
            Circunferencia c = new CircunferenciaImpl2(centro,3.0);  
            System.out.println("c: "+ c);  
        }catch(Exception e){  
            System.out.println("Excepción capturada:\n " + e);  
        }  
    }  
}
```



Captura de Excepciones try-catch

2024/25

Ejemplo:

Captura del tipo Circunferencia (continuación):

```
...
System.out.println("\nTEST del Constructor radio 0");
try{
    Circunferencia c = new CircunferenciaImpl2(centro,0d);
    System.out.println("c: "+ c);
}catch(Exception e){
    System.out.println("Excepción capturada:\n " + e);
}

System.out.println("\nTEST del Constructor radio negativo");
try{
    Circunferencia c = new CircunferenciaImpl2(centro,-2.5);
    System.out.println("c: "+ c);
}catch(Exception e){
    System.out.println("Excepción capturada:\n " + e);
}
```




2024/25

Captura de Excepciones try-catch

Ejemplo:

Captura del tipo Circunferencia (continuación):

```
private static void testSetRadio() {
    Punto centro=new Punto(1.0,-1.0);
    Circunferencia c = new CircunferenciaImpl1(centro,3.0);
    System.out.println("\nTEST setRadio con radio correcto");
    try {
        c.setRadio(10d);
        System.out.println("c: " + c);
    }catch(Exception e) {
        System.out.println("Excepción capturada:\n " + e);
    }

    System.out.println("\nTEST setRadio con radio cero");
    try {
        c.setRadio(0d);
        System.out.println("c: " + c);
    }catch(Exception e) {
        System.out.println("Excepción capturada:\n " + e);
    }
}
```



2024/25

Captura de Excepciones try-catch

Ejemplo:

Captura del tipo Circunferencia (continuación):

```
...
System.out.println("\nTEST setRadio con radio negativo");
try {
    c.setRadio(-20d);
    System.out.println("c: " + c);
} catch (Exception e) {
    System.out.println("Excepción capturada:\n " + e);
}
}
```