Programación III

- ✔ Proceso Disciplinado de Desarrollo

- **Precondiciones:** condiciones para que una rutina funcione correctamente. El objeto que realiza una invocación a un método garantiza los pre-requisitos
- **Postcondiciones:** describen el efecto de la rutina. El método invocado garantiza proveer el resultado previsto.
- **Invariante:** estado o situación que no cambia dentro de un contexto o porción de código. Hay invariantes de *estado* e invariantes de *ciclo*.

El invariante de una clase (de estado)

El **invariante de una clase** se expresa mediante conjunto de aserciones que indican las propiedades que en todo momento deben cumplir las instancias de esa clase, y que pueden utilizarse como suposiciones dentro de todos los métodos.

Esta compuesto por:

1. restricciones sobre los valores que pueden tomar los atributos de la clase
2. restricciones sobre los valores que pueden tomar los objetos hacia los cuales hay una asociación
3. relaciones entre los atributos y/o los objetos con los cuales se relaciona.

El invariante de bucle (o ciclo)

Un invariante de bucle es una condición [entre las variables del programa] que es necesariamente verdadero inmediatamente antes e inmediatamente después de cada iteración de un bucle.

El fallo de un invariante de bucle significa que el algoritmo se ha perdido. Tomemos como ejemplo el caso de estar buscando una palabra en un diccionario por el método binario. El algoritmo consiste en abrir el diccionario por la mitad y determinar en qué mitad está la palabra, y así recurrentemente con la mitad de la mitad de la mitad, etc. En todo momento debe ser cierto que la palabra que buscamos esté en la fracción de diccionario que estamos buscando; es decir, que sea posterior a la primera palabra de la fracción, y anterior a la última.

Precondición, Invariante, Postcondición

En todo método se puede suponer que:

- al comienzo de su ejecución se cumplen todas las afirmaciones que aparecen en el invariante de clase y en la precondición del contrato,
- después de haber sido ejecutado sobre un objeto, éste cumple todas las afirmaciones del invariante y de la postcondición.

EJEMPLO SIMPLE

```
public class Cuenta {
    private static int ultimoCodigo = 0;
    private int codigo;
    private double saldo;
    private final Persona titular;
    private EstadoCuenta estado;

    public Cuenta(Persona persona, int saldoInicial) {
        codigo = ++ultimoCodigo;
        saldo = saldoInicial;
        titular = persona;
        estado = EstadoCuenta.OPERATIVA;
    }
    public void ingreso(double cantidad) {

        saldo = saldo + cantidad;
    }
    ...
}
```

Clase Cuenta:

- El estado de una cuenta es correcto si tiene un titular (referencia no nula) → invariante
- El método ingreso() incrementa el saldo de la cuenta → postcondición
- El parámetro del método ingreso() debe ser siempre positivo → precondición
- El método ingreso() sólo se puede ejecutar en estado OPERATIVA → precondición
- Cada cuenta tiene un código diferente.

EJEMPLO SIMPLE

```
public void ingreso(double cantidad)
{
    [precondición: cantidad >= 0]
    [precondición: Estado de cuenta Operativo]
    saldo = saldo + cantidad;
    [postcondición: saldo = saldo + cantidad]
    [invariante: titular != null]
}
```

Aserciones y Excepciones

Java proporciona dos herramientas para tratar con la **corrección** y **robustez** del código:

- **Aserto o Aserciones:** condición que debe cumplirse en el código.
- **Excepción:** mecanismo para notificar un error durante la ejecución.

Tanto las precondiciones, postcondiciones e invariantes pueden expresarse con aserciones.

Aserciones

Una Aserción es una condición lógica insertada en el código Java, de ideas o condiciones que se asumen son ciertas. El sistema se encarga de comprobarlas y avisar mediante una excepción en caso de que no se cumplan.

Generalmente se usan para verificar valores de las variables en ciertos puntos del programa

Se aconseja su uso durante la etapa de desarrollo.

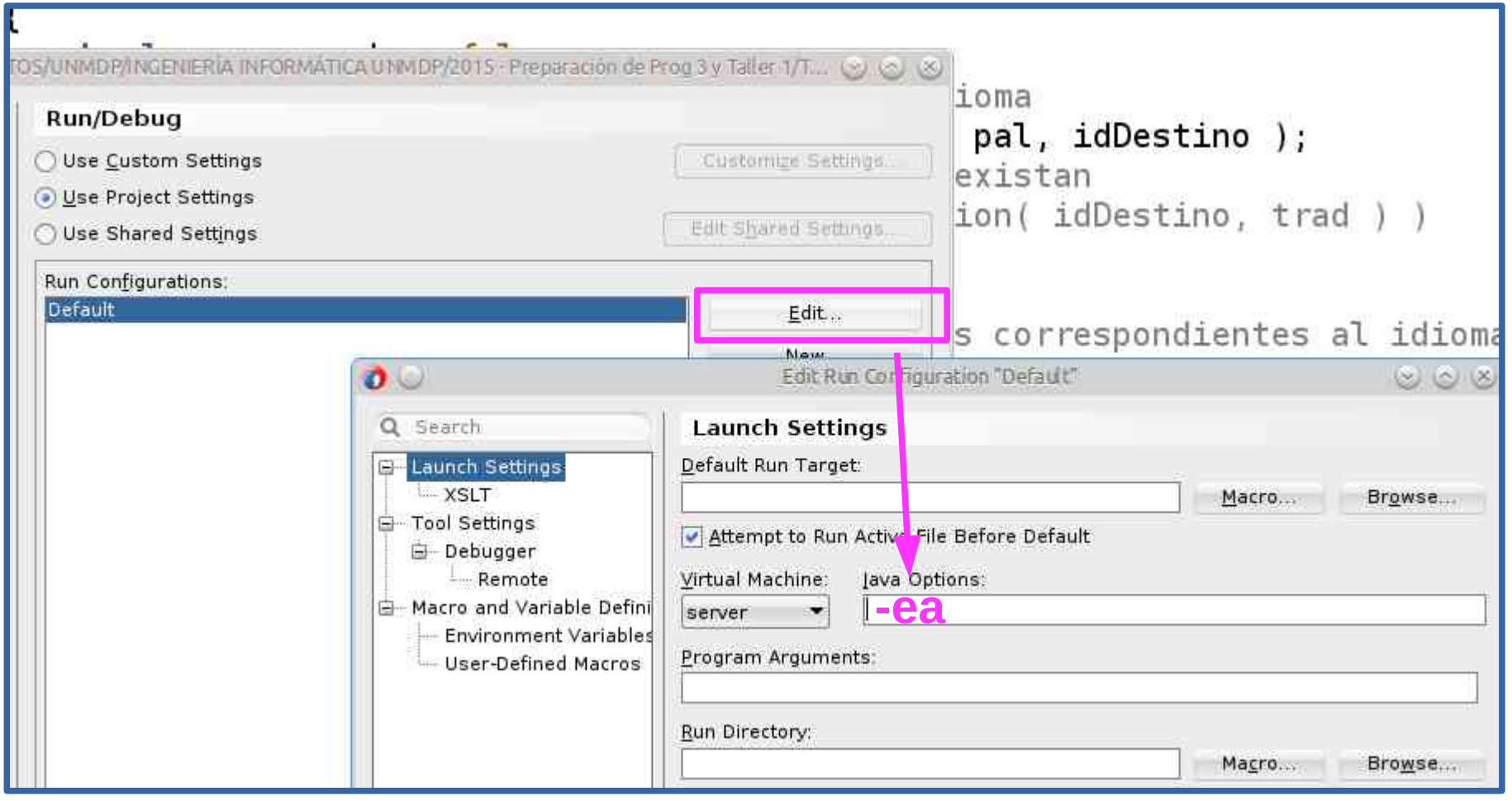
Aserciones

Las aserciones admiten dos tipos de sintaxis:

```
assert <expresion_booleana>;
assert <expresion_booleana> : <expresion_detallada>;
```

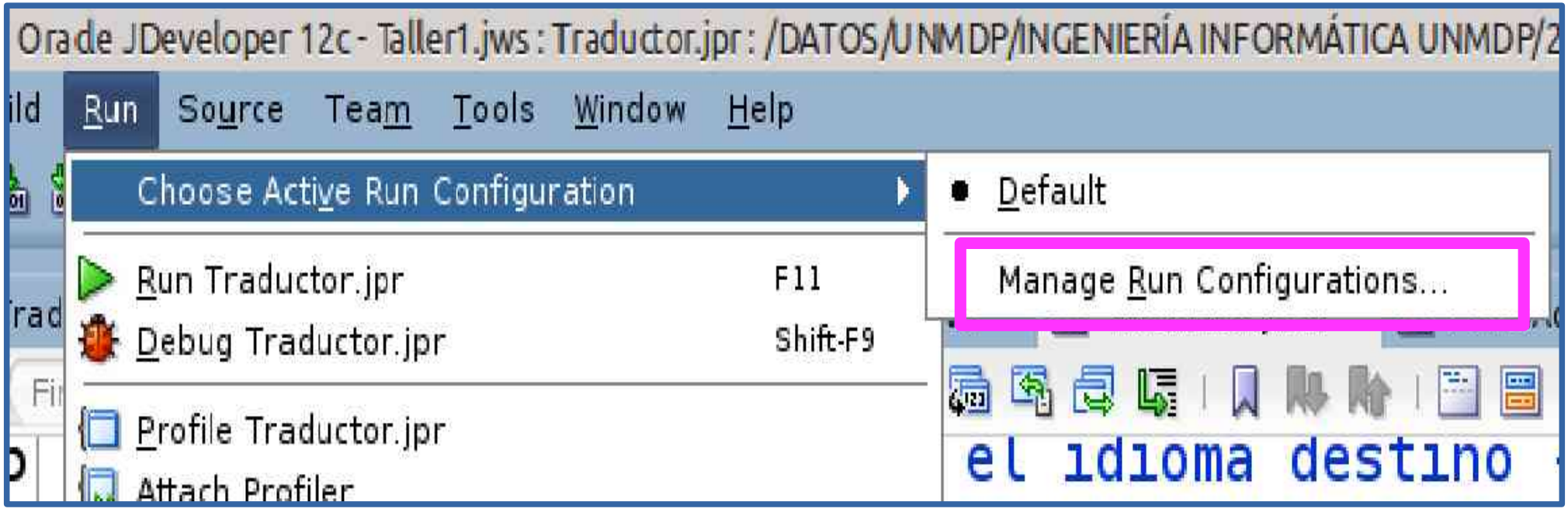
- En cualquier caso, si la expresión booleana evaluada es falsa, se genera un error de aserción (**AssertionError**). Este tipo de error no debería capturarse y **el programa debería finalizar de forma anormal**.
- En caso de utilizarse el segundo formato, la segunda expresion (que puede ser de cualquier tipo) será convertida al tipo String y se utilizará para complementar el mensaje a mostrar por pantalla al momento de producirse el AssertionError.
- Se pueden **activar o desactivar** de manera sencilla. Se agrega la opción -ea al comando de ejecución: `java -ea MiClase`

Aserciones – activar en JDeveloper



Aserciones – activar en JDeveloper

Para activarlo en el entorno de desarrollo:



Pre y postcondiciones del método ingreso():

```
public void ingreso(double cantidad)
{
    assert cantidad >= 0: “cantidad negativa”;
    assert estado == EstadoCuenta.Operativa:“estado incorrecto”;
    saldo = saldo + cantidad;
    assert saldo = oldSaldo + cantidad: “fallo postcondición”;
    /FALTA DEFINIR EL INVARIANTE
}
```

```
public void ingreso(double cantidad)
{
  [precondición: cantidad >= 0]
  [precondición: Estado de cuenta Operativo]
   saldo = saldo + cantidad;
  [postcondición: saldo = saldo + cantidad]
  [invariante: titular != null]
}
```

Postcondición:

- El código anterior **no compila**, ya que no es posible expresar con un aserto el antiguo valor de un atributo (**`old saldo`**).
- Para expresar correctamente el aserto deberíamos declarar una variable local para almacenar el antiguo valor del atributo.

Pre y postcondiciones del método ingreso():

```
public void ingreso(double cantidad)
{
    assert cantidad >= 0: “cantidad negativa”;
    assert estado == EstadoCuenta.Operativa:“estado incorrecto”;
    double oldSaldo = saldo;
    saldo = saldo + cantidad;
    assert saldo = oldSaldo + cantidad: “fallo postcondición”;
    assert invariante(): "Fallo invariante";
}
```

```
private boolean invariante()
{
    return titular != null;
}
```

Invariante:

- Podría definirse en un método privado que sea comprobado al final de cada método y constructor

Inconvenientes de los asertos

- Los asertos en Java son pobres.
  - Por ejemplo, no podemos establecer en un aserto el antiguo valor de un atributo (“*se neceista oldSaldo*”).
- No permite utilizar cuantificadores lógicos ni condiciones globales a una clase.
  - Por ejemplo, “cada cuenta debe tener un código único”.
- Un aserto sólo es comprobado durante la ejecución del código.
- Los asertos no se heredan.
- Por defecto, los asertos están desactivados.
  - Se activan con el parámetro -ea de la máquina virtual.