

# **Depuración y realización de pruebas**

# Depuración y realización de pruebas

- Herramientas de depuración
- Análisis de código
- Casos de prueba
- Pruebas unitarias

# Herramientas de depuración

- Nos permite identificar y corregir errores de programación mediante la ejecución controlada del software
- El IDE nos provee de un depurador o debugger
  - Ejecución paso a paso del programa
  - Estado de los métodos, variables, objetos, pila de ejecución
  - Puntos de interrupción

# Herramientas de depuración

- Puntos de interrupción / Puntos de ruptura
  - Puntos de control situados en líneas concretas del código fuente
  - El depurador detiene la ejecución del programa al llegar a cada uno de los puntos de interrupción definidos en el IDE
  - Sólo se tendrán en cuenta al ejecutar el programa en modo “depuración”
    - Si ejecutamos normalmente el programa no se detendrá y obviará los puntos de ruptura definidos

# Herramientas de depuración

- Definición de punto de ruptura en Eclipse:
  - Doble click junto a la línea
  - Aparece un pequeño círculo
  - Ejemplo. Breakpoint en línea 43:

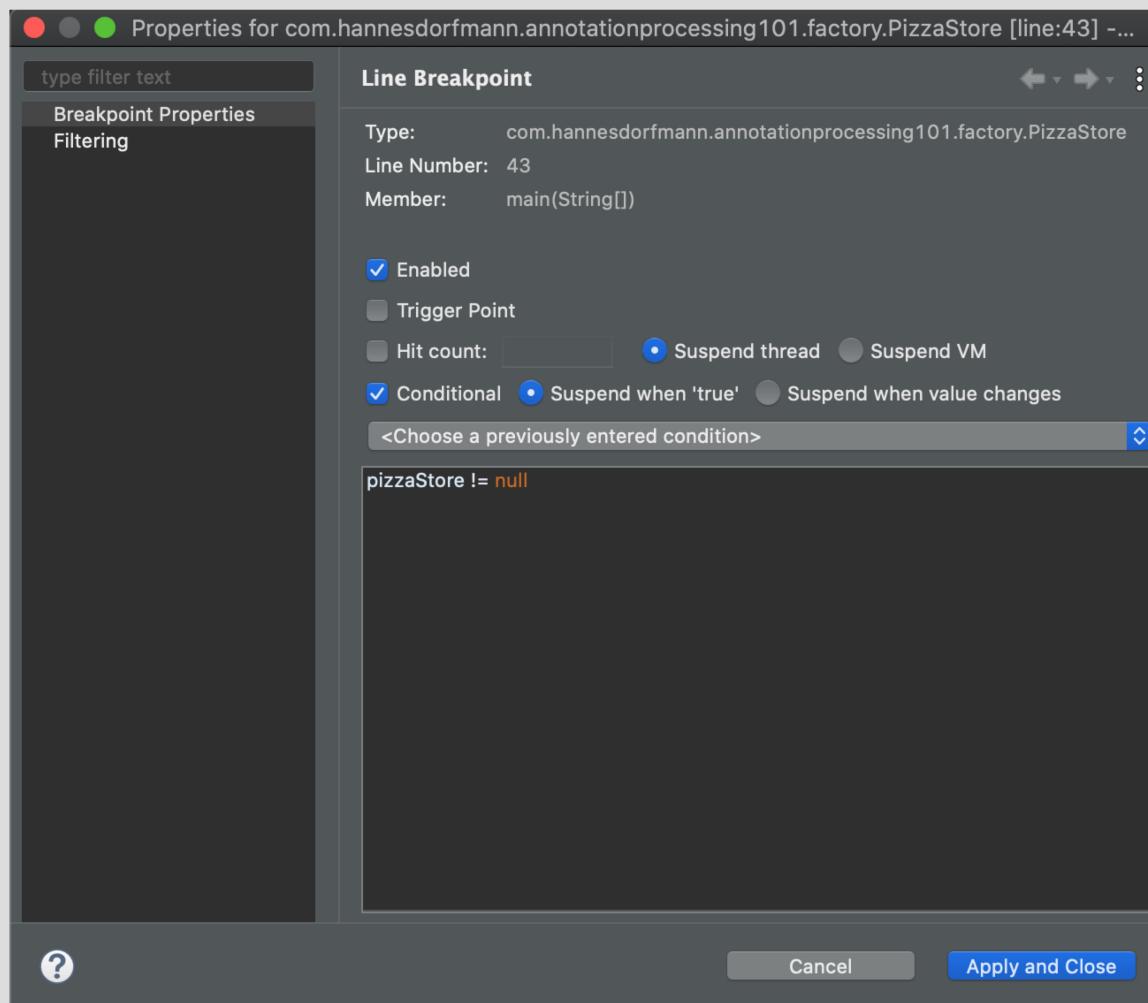
```
41 public static void main(String[] args) throws IOException {  
42     PizzaStore pizzaStore = new PizzaStore();  
● 43     Meal meal = pizzaStore.order(readConsole());  
44     System.out.println("Bill: $" + meal.getPrice());  
45 }
```

# Herramientas de depuración

- Puntos de interrupción / Puntos de ruptura
  - Se pueden definir condiciones de interrupción del programa
    - Normalmente, querremos que el programa se detenga cuando la condición sea *true*
  - Se pueden deshabilitar los puntos de ruptura
  - Para ver las propiedades de un punto de ruptura en eclipse, click derecho sobre el breakpoint

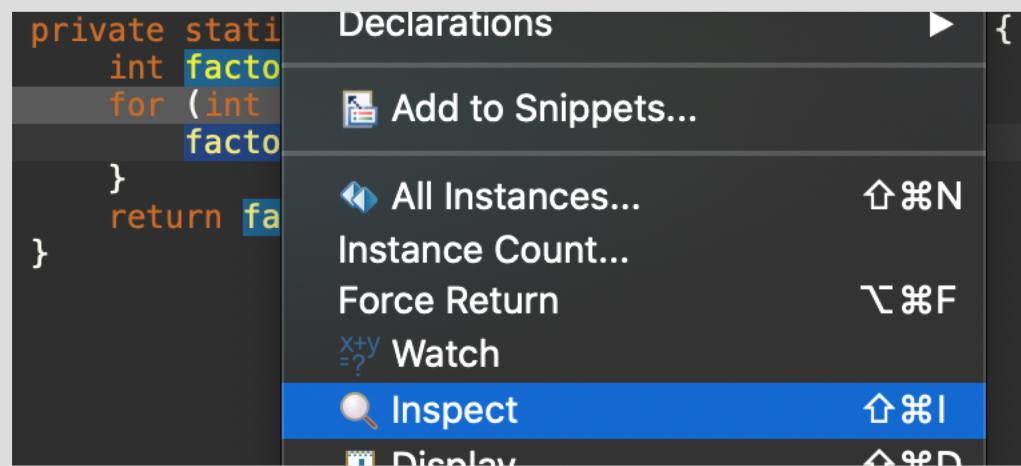
# Herramientas de depuración

- Puntos de interrupción / Puntos de ruptura



# Herramientas de depuración

- Inspecciones
  - Podemos conocer el valor de las variables mediante la opción de “Inspect”
    - En eclipse, con el depurador parado en un punto de ruptura, seleccionamos una variable y podemos inspeccionarla con la opción que aparece en su menú contextual:



# Herramientas de depuración

- Ejercicio
  - Crea una aplicación que pasado un número como argumento sea capaz de calcular su serie de Fibonacci:
    - Recordatorio:  $F_1 = F_2 = 1$  y  $F_n = F_{n-1} + F_{n-2}$
  - Utiliza la herramienta de depuración para
    - Definir condiciones en los puntos de interrupción
    - Inspeccionar el valor de una variable (un cálculo intermedio)

# Depuración y realización de pruebas

- Herramientas de depuración
- **Análisis de código**
- Casos de prueba
- Pruebas unitarias

# Análisis de código

- Análisis de código
  - Warnings o avisos
    - Permite continuar la compilación y generar un ejecutable
  - Errores
    - No permite continuar la compilación y generar un ejecutable

# Análisis de código

- Análisis de código
  - SonarQube : herramienta de análisis de código que permite recopilar información de varias métricas realizando un informe evaluador del propio código
  - Métricas
    - Líneas de código,
    - complejidad ciclomática,
    - Código repetido
    - Cobertura de tests
    - Violaciones de buenas prácticas

# Análisis de código

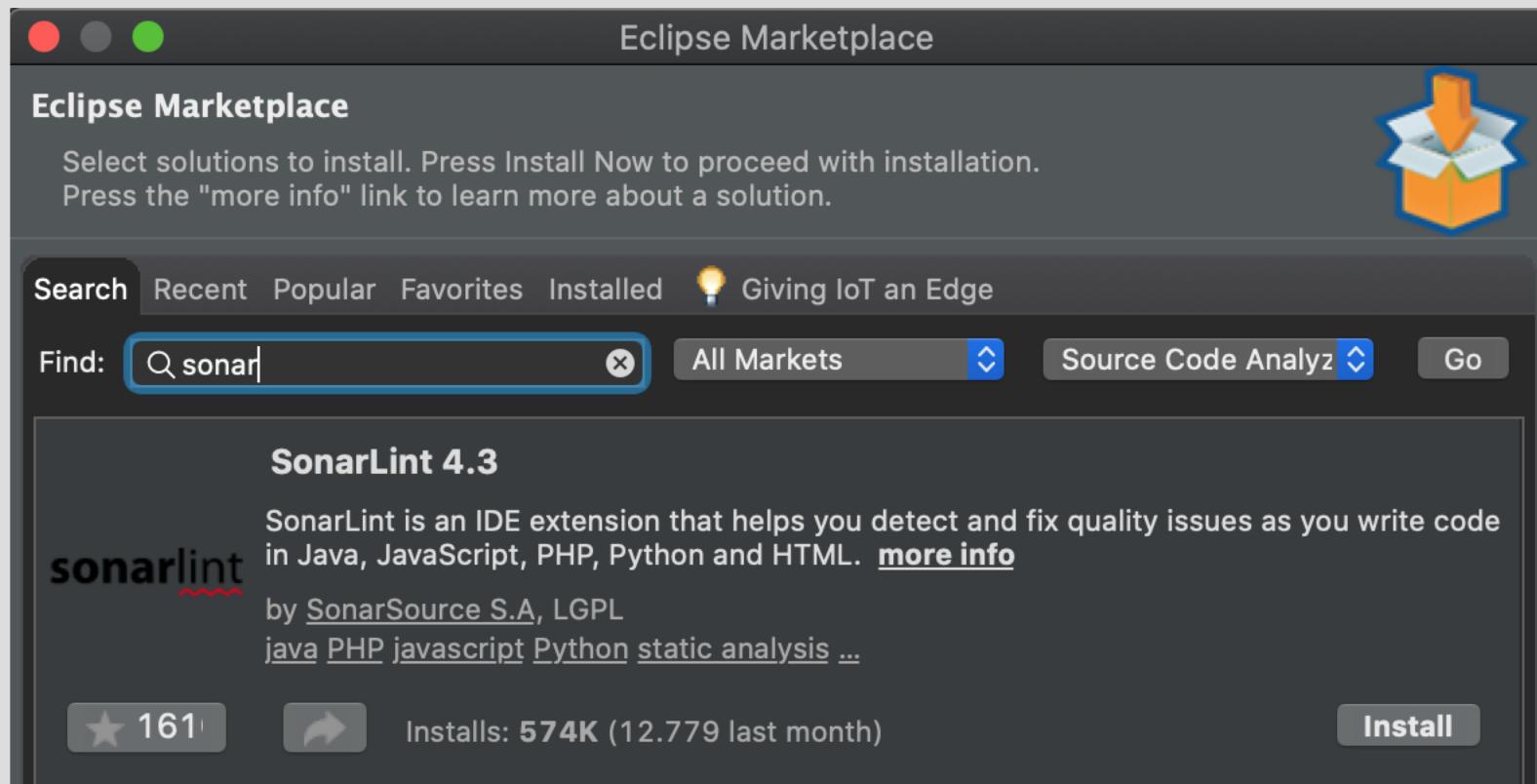
- Sonar
  - Estima la deuda técnica
    - Esfuerzo en tiempo y dinero de mantenimiento de un software de mala calidad
    - Mala imagen frente al cliente
    - Malestar en el equipo
  - La deuda técnica surge por la presión sobre los desarrolladores.
    - Mala planificación o estimación

# Análisis de código

- Sonar
  - Idealmente, el análisis del código se debe hacer
    - Con cada push al repositorio
    - De manera periódico
    - Corrigiendo errores previos
  - Se puede hacer un análisis del código también en local
    - Plugin de eclipse: SonarLint, centrado para trabajar en el IDE

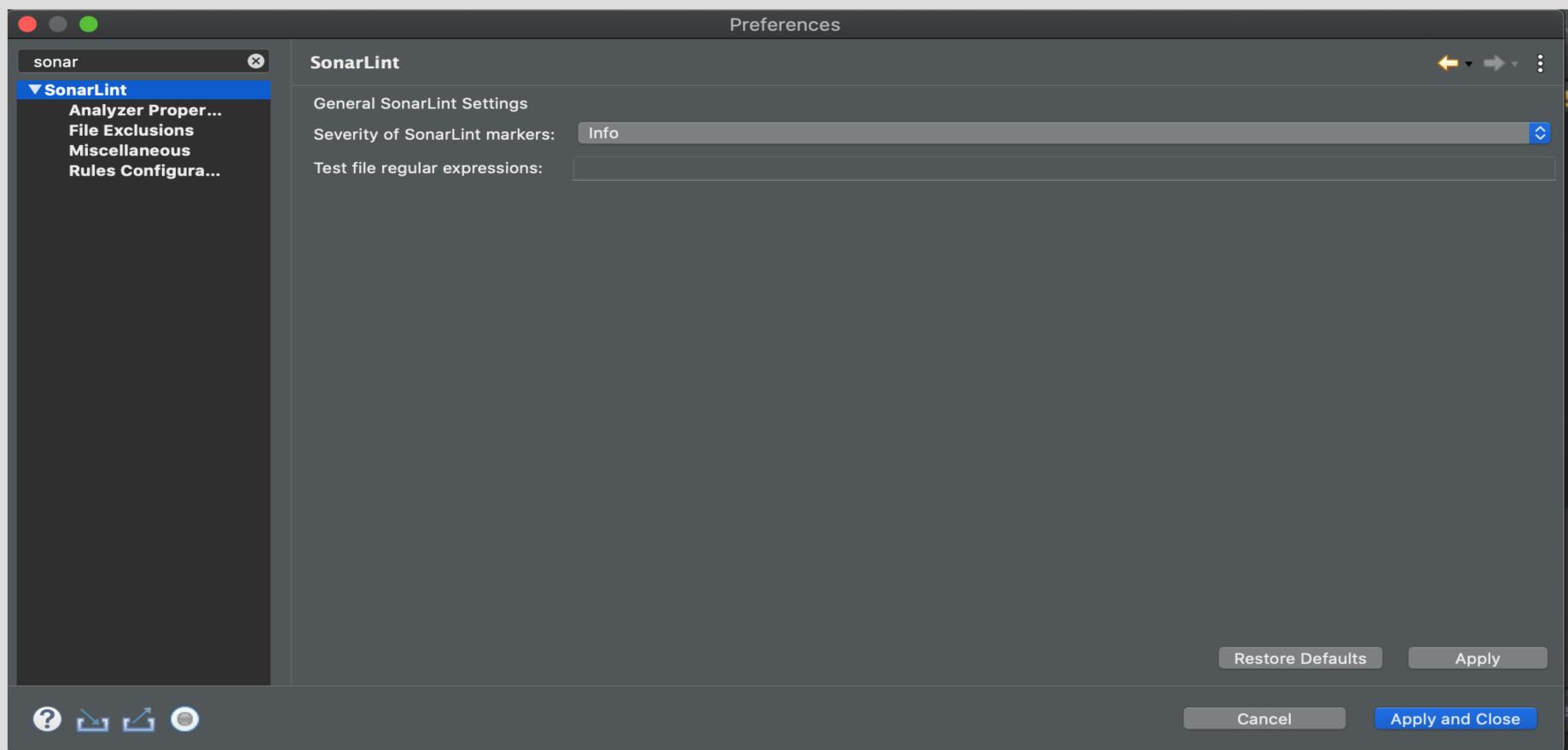
# Análisis de código

- SonarLint
  - Instalación desde Eclipse Marketplace



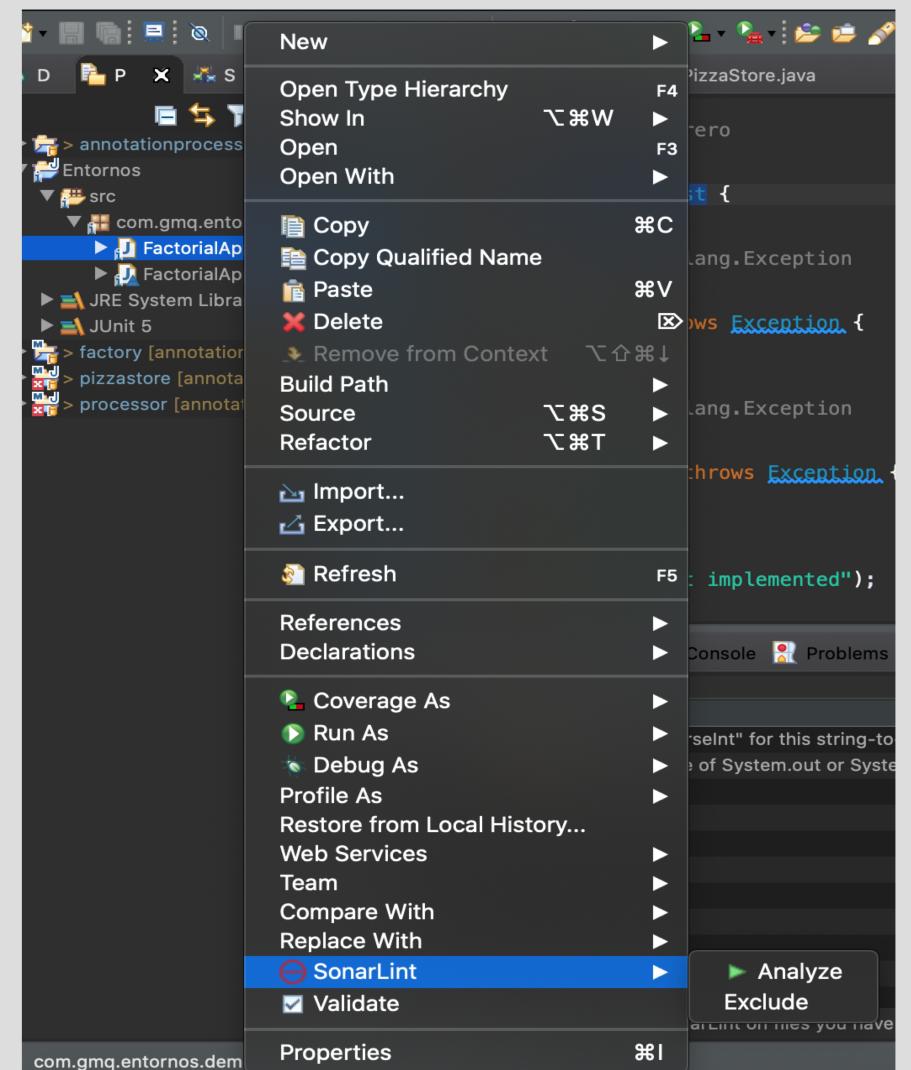
# Análisis de código

- SonarLint
  - Preferencias y configuración del plugin



# Análisis de código

- SonarLint
  - Análisis desde menú contextual

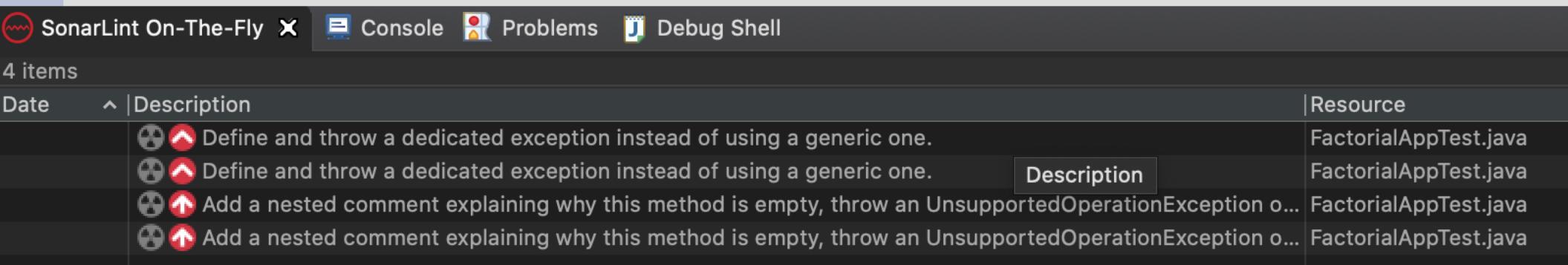


# Análisis de código

- SonarLint

```
public static void main(String[] args) {  
    int originalNumber = Integer.valueOf(args[0]).  
    int factorial = calculateFactorial(originalNumber),  
        System.out.println("El factorial es "+factorial);  
}
```

- Menú: Show view → sonar



The screenshot shows the SonarLint On-The-Fly interface integrated into a Java IDE. The top navigation bar includes tabs for SonarLint On-The-Fly, Console, Problems, and Debug Shell. Below the navigation bar, a message indicates there are 4 items. A table displays the analysis results:

Date	Description	Resource
	Define and throw a dedicated exception instead of using a generic one.	FactorialAppTest.java
	Define and throw a dedicated exception instead of using a generic one.	FactorialAppTest.java
	Add a nested comment explaining why this method is empty, throw an UnsupportedOperationException o...	FactorialAppTest.java
	Add a nested comment explaining why this method is empty, throw an UnsupportedOperationException o...	FactorialAppTest.java

# Depuración y realización de pruebas

- Herramientas de depuración
- Análisis de código
- **Casos de prueba**
- Pruebas unitarias

# Casos de prueba

- Pensados para asegurar el correcto funcionamiento de una aplicación
- Deben abarcar
  - el máximo código posible
  - La mayor cantidad de casos (alternativas) posibles
- Deben programarse a la vez que el código fuente o incluso antes
  - TDD: Test Driven Development
- El código de las pruebas no debe descuidarse

# Casos de prueba

- Pruebas de caja blanca
  - Se centran en el **cómo**
- Pruebas de caja negra
  - Se centran en el **qué**

# Casos de prueba

- Pruebas de caja blanca
  - Tienen en cuenta los parámetros de entrada y la salida
  - Valida el correcto funcionamiento y también el control de errores
  - Pruebas de cada método *public*
  - En Java, framework *JUnit*

# Casos de prueba

- Pruebas de caja negra
  - Enfocadas solamente en las entradas y salidas del sistema
  - Se toma el sistema como un ente global
  - La funcional se verifica sin tomar en cuenta la estructura interna de código, detalles de implementación o escenarios de ejecución internos en el software
  - Pruebas funcionales

# Casos de prueba

- Casos de prueba
  - Análisis de valores límite: casos de prueba son diseñados basándose en los valores límite del método que se testea
  - Por ejemplo:
    - Si un método acepta valores comprendidos entre 0 y 100, probar con los valores -1, 0, 1, 99, 100 y 101

# Casos de prueba

- Casos de prueba
  - Pruebas de rendimiento o estrés: se somete al sistema a un gran número de peticiones concurrentes para comprobar su funcionamiento
    - Se comprueba:
      - Si el sistema es capaz de aceptar peticiones
      - Los tiempos de respuesta
      - Uso de memoria
      - Uso de CPU

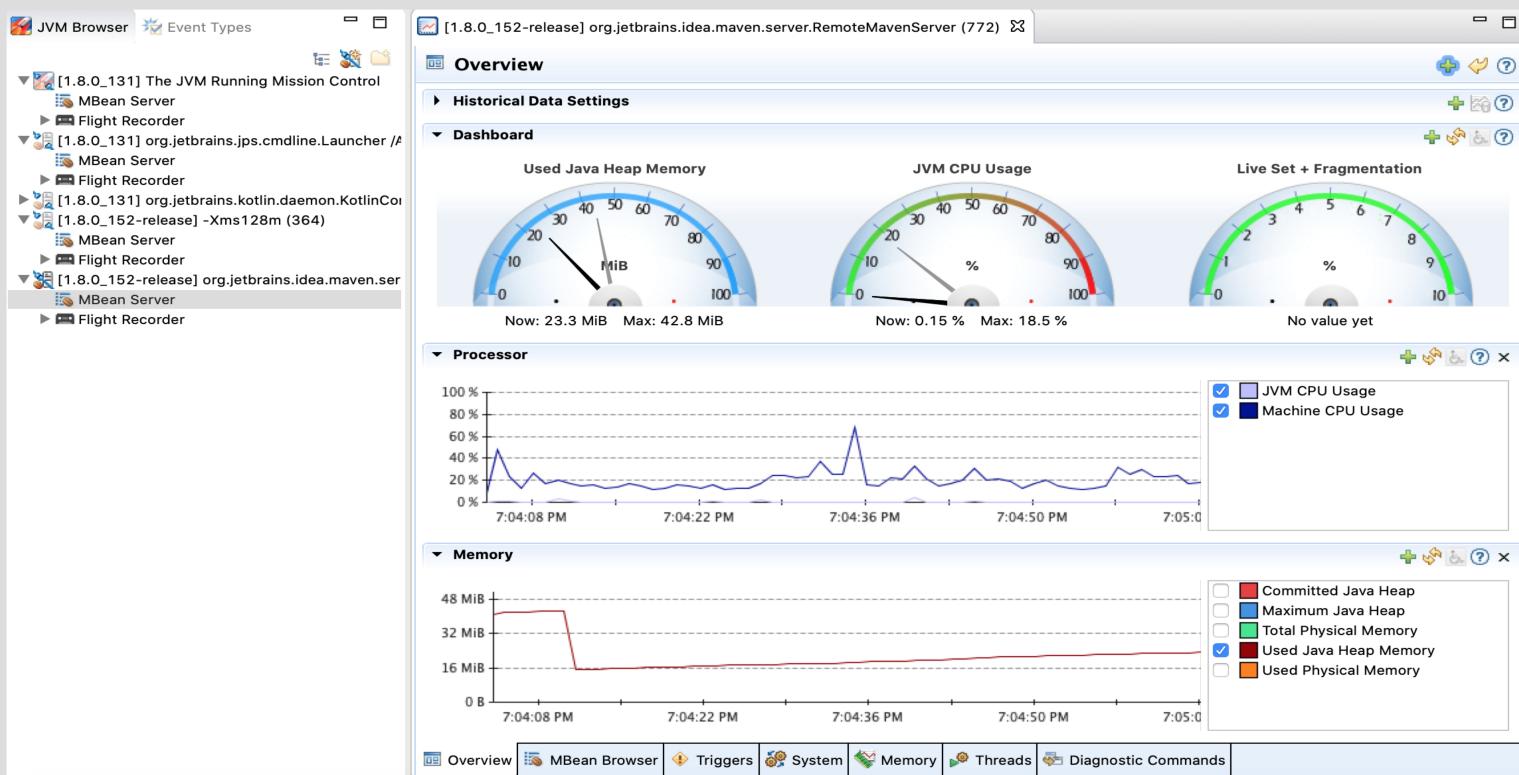
# Casos de prueba

- Pruebas de rendimiento o estrés
  - Jmeter
  - Gatling



# Casos de prueba

- Pruebas de rendimiento o estrés
  - Java Mission Control:
    - Interfaz gráfica: JMC Client (JAVA\_HOME\bin\jmc)



# Casos de prueba

- Pruebas de aceptación
  - Se realizan en las últimas etapas del desarrollo antes de lanzar una RELEASE
  - Determinar si cumplen con las necesidades y/o requerimientos de las empresas y sus usuarios
  - Manuales o automatizadas
  - Se pueden a Versiones Canarias (*Canary version*): desarrollos específicos en subconjuntos de usuarios (preseleccionados de acuerdo a un criterio o aleatoriamente).
    - El nombre de Canary Version proviene del uso práctico de usar canarios en las minas de carbón para detectar presencia de gases tóxicos. El daño controlado era que murieran las aves y no los seres humanos. □

# Casos de prueba

- Pruebas de aceptación
  - Selenium
    - Entorno de pruebas de software para aplicaciones basadas en la web
    - Emula la interacción con navegadores web
    - Librería para escribir tests en java



# Depuración y realización de pruebas

- Herramientas de depuración
- Análisis de código
- Casos de prueba
- **Pruebas unitarias**

# Pruebas unitarias

- Son pruebas de caja blanca
- El propio desarrollador es quien las codifica
  - Aunque no es lo ideal
- Fundamentales
  - Forman parte de una entrega
- Automatizables
  - Forman parte de una entrega

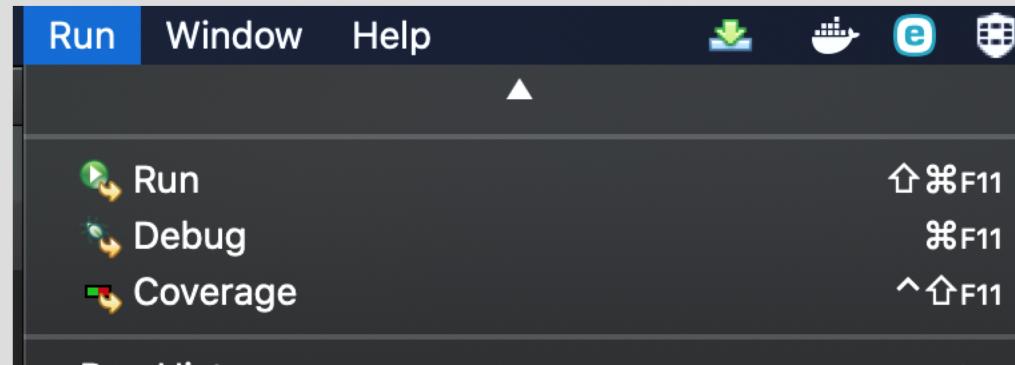
# Pruebas unitarias

- Pruebas individuales de cada método publico
- Conocemos los valores de entrada y cuál debería ser la salida
- Misma estructura de paquetes que el código fuente
- Framework Junit 5
  - Integrado en eclipse



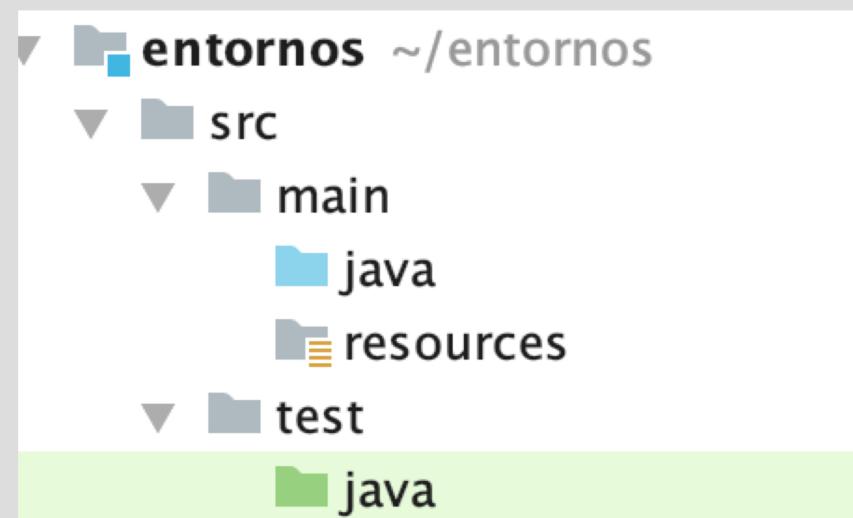
# Pruebas unitarias

- Cobertura: porcentaje de código fuente testeado con las pruebas unitarias
- Eclipse: plugin EclEmma
  - Añade un nuevo modo de lanzamiento “Coverage” desde el menú “Run”



# Pruebas Unitarias

- Directorio de los tests
  - Junto al código fuente



# Pruebas unitarias

- Definición de un test
  - Convención: la clase de test se llamará igual que la clase testada con el sufijo Test.
    - Ej: Calculadora.java → CalculadoraTest.java
  - Convención de los métodos setUp y tearDown
    - setUp: inicialización del test
    - tearDown: finalización del test

# Pruebas unitarias

- setUp
  - @BeforeEach / @BeforeAll
- tearDown
  - @AfterEach / @AfterAll

```
@BeforeEach
void setUp() {
    //inicialización de variables
}

@AfterEach
void tearDown() {
    //finalización conexiones, cierre de ficheros ...
}
```

# Pruebas unitarias

- Junit: convención de nombres
  - El nombre del test debería explicar lo que hace, lo que quiere probar
    - Puede empezar por *should\_*  
P.ej: *should\_read\_configuration\_file*
    - Si se prueba que se lanzar una excepción, podría empezar por *throws\_*  
P.ej: *throws\_exception\_when\_reads\_file*
    - Given[Input]When[DoSthng]Then[Result]  
P.ej:  
*GivenConfigFileWhenStartsThenFileConfig*

# Pruebas unitarias

- Aserciones
  - Javadoc clase de junit Assertions
    - fail
    - assertAll
    - assertTrue / assertFalse
    - assertEquals
    - assertThrows

# Pruebas unitarias

- Ejercicio
  - Tomando como código fuente el ejercicio de cálculo de la serie de Fibonacci, crea los tests necesarios con Junit
    - Crea el directorio de los tests
    - Utiliza las anotaciones vistas en clase

# Bibliografía

- Entornos de desarrollo, Editorial RA-MA
- <https://www.adictosaltrabajo.com/2014/10/08/eclipse-so>
-