CAPTTON
IT CONSULTING

Excepciones

Cuando algo hace *Kaboom* hay que estar preparado

Excepciones

- Las excepciones ocurren cuando sucede algo inesperado dentro de la lógica de nuestro programa por ejemplo un pasaje incorrecto de argumentos, una falla en la red, un archivo no encontrado, etc.
- Sintaxis:

```
try{
  //Bloque codigo
}
catch{
  //Bloque codigo
}
finally{
  //Bloque codigo
}
```

Try / Catch / Finally

- Try – Se utiliza para encerrar código que potencialmente podría dar error
- Catch – En caso de que dicho código dé error, esta sentencia tomará el error y ejecutará el código que se defina en el bloque.
  - Se puede definir un catch genérico (tomará cualquier error que suceda) o un definir un tipo de error específico. También se pueden definir multiples catch
- Finally – Esta sentencia se ejecutará siempre que este definida, sin importar cómo terminó el try (Si dio o no error)

Excepciones

- Si una excepción no es manejada en el momento que se lanza, dicha excepción subirá al método que invocó el código que genero la excepción hasta que alguien lo “agarre” o hasta que llegue a finalizar el programa abruptamente
- Podemos implementar nuestras propias excepciones heredando de la clase Exception
- Cuando tenemos excepciones implementadas por nosotros, vamos a querer poder lanzarlas. Para esto se utiliza la palabra reservada ‘Throw’

Aserción

Assertions

- La sentencia ‘Assert’ se utiliza para probar una condición
- Si la condición devuelve verdadero, no sucede nada
- Si la condición devuelve falso, el Assert falla.

Pruebas Unitarias

CAPTON
IT CONSULTING

Pruebas Unitarias

- Una prueba unitaria es una forma de probar el correcto funcionamiento de un módulo de código.
- Sirve para asegurar que cada uno de los módulos funcione correctamente por separado.
- El objetivo de las pruebas unitarias es aislar cada parte del programa y mostrar que las partes individuales son correctas.

Pruebas Unitarias - Características

- Automatizable
  - No debería requerirse una intervención manual.
- Completas
  - Deben cubrir la mayor cantidad de código.
- Reutilizables
  - No se deben crear pruebas que sólo puedan ser ejecutadas una sola vez
- Independientes
  - La ejecución de una prueba no debe afectar a la ejecución de otra.
- Profesionales
  - Las pruebas deben ser consideradas igual que el código, con la misma profesionalidad, documentación, etc.

Pruebas Unitarias - Ventajas

- Fomentan el cambio
  - Facilitan la refactorización, puesto que permiten hacer pruebas sobre los cambios y así asegurarse de que los nuevos cambios no han introducido errores.
- Separación de la interfaz y la implementación
  - Dado que la única interacción entre los casos de prueba y las unidades bajo prueba son las interfaces de estas últimas, se puede cambiar cualquiera de los dos sin afectar al otro.
- Los errores están más acotados y son más fáciles de localizar
  - La prueba unitaria que falle nos ayuda a saber en que funcionalidad implementada se está produciendo la falla.

Pruebas Unitarias – Tipos

- Pruebas de Caja Blanca
  - Conociendo el código y siguiendo su estructura lógica, se pueden diseñar pruebas destinadas a comprobar que el código hace correctamente lo que se espera
- Pruebas de Caja Negra
  - Pruebas funcionales
  - Se parte de los requisitos funcionales, a muy alto nivel
  - Las pruebas se aplican sobre el sistema sin conocer como esta constituido por dentro
  - Se tiene en cuenta los datos de entrada y su resultado

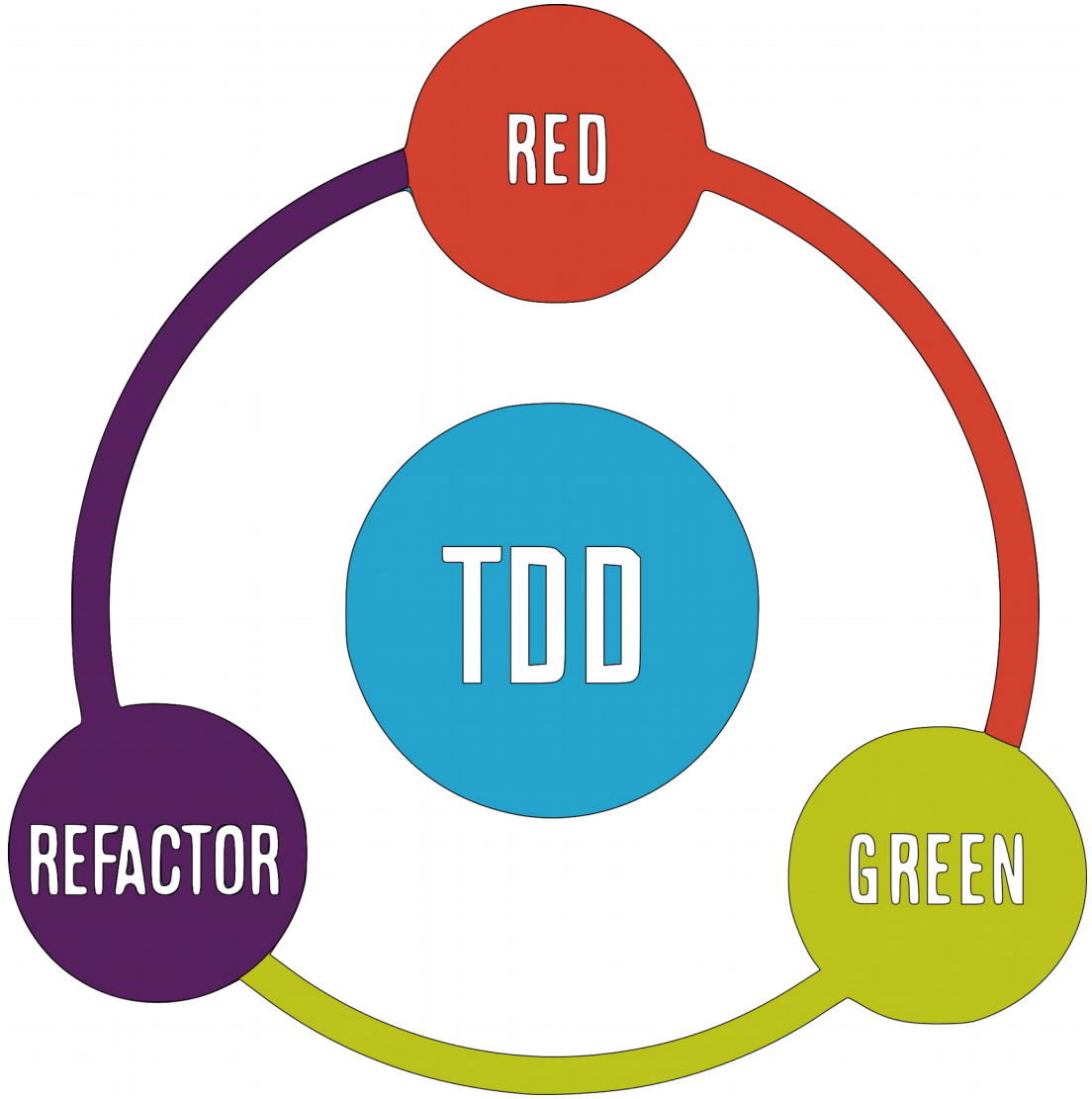
CAPTTON
IT CONSULTING

Desarrollo guiado por pruebas

AKA Test Driven Development (TDD)

TDD

- Proceso de desarollo de software que se basa en la repetición de un ciclo de desarrollo reducido



CAPTTON
IT CONSULTING

JUnit

Pruebas Unitarias

CAPTTON
IT CONSULTING

JUnit – ¿Que es?

- JUnit es un framework open source que se utiliza para escribir y ejecutar pruebas unitarias
- Provee Anotaciones para identificar los métodos utilizados para las pruebas
- Provee Aserciones para probar los resultados esperados
- Provee ‘Test runners’ para ejecutar pruebas