SAP Customer Experience

Test-Driven Development

We will learn about:

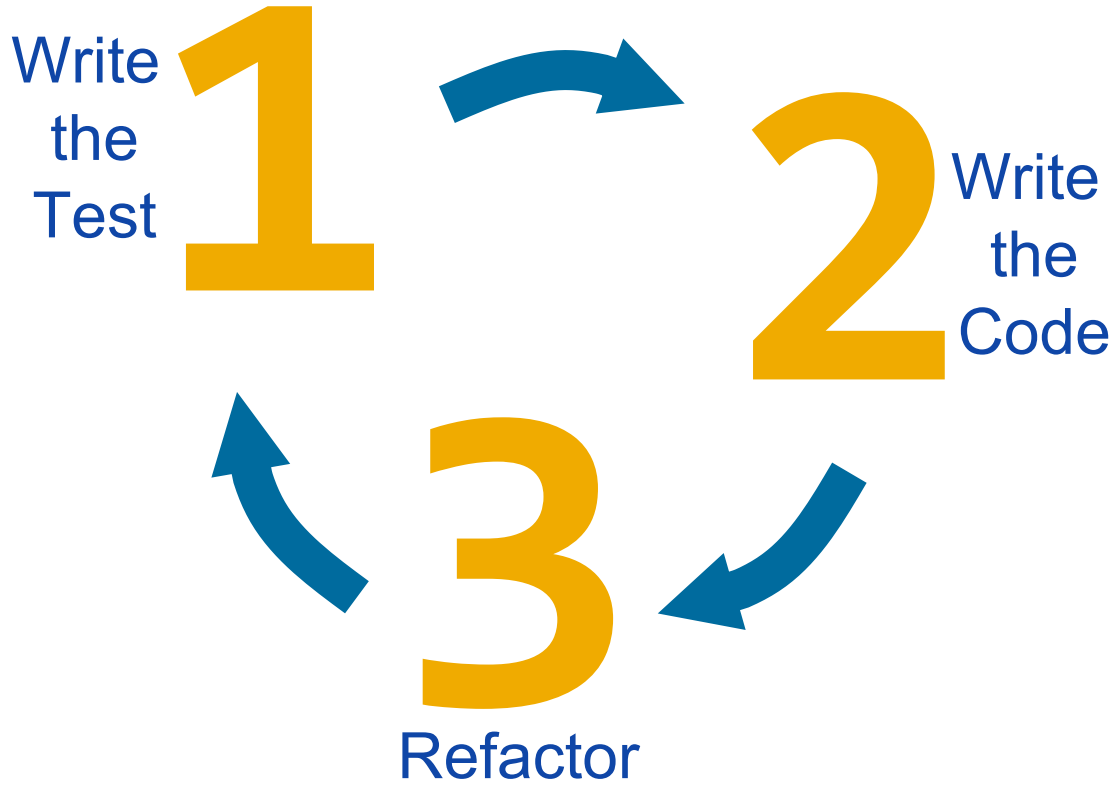
- Concept
- JUnit environment

Concept

Test Driven Development (TDD)

Developing software by developing the tests first.

Introduced by Kent Beck as part of his work on Extreme Programming (XP).



Some key TDD principles

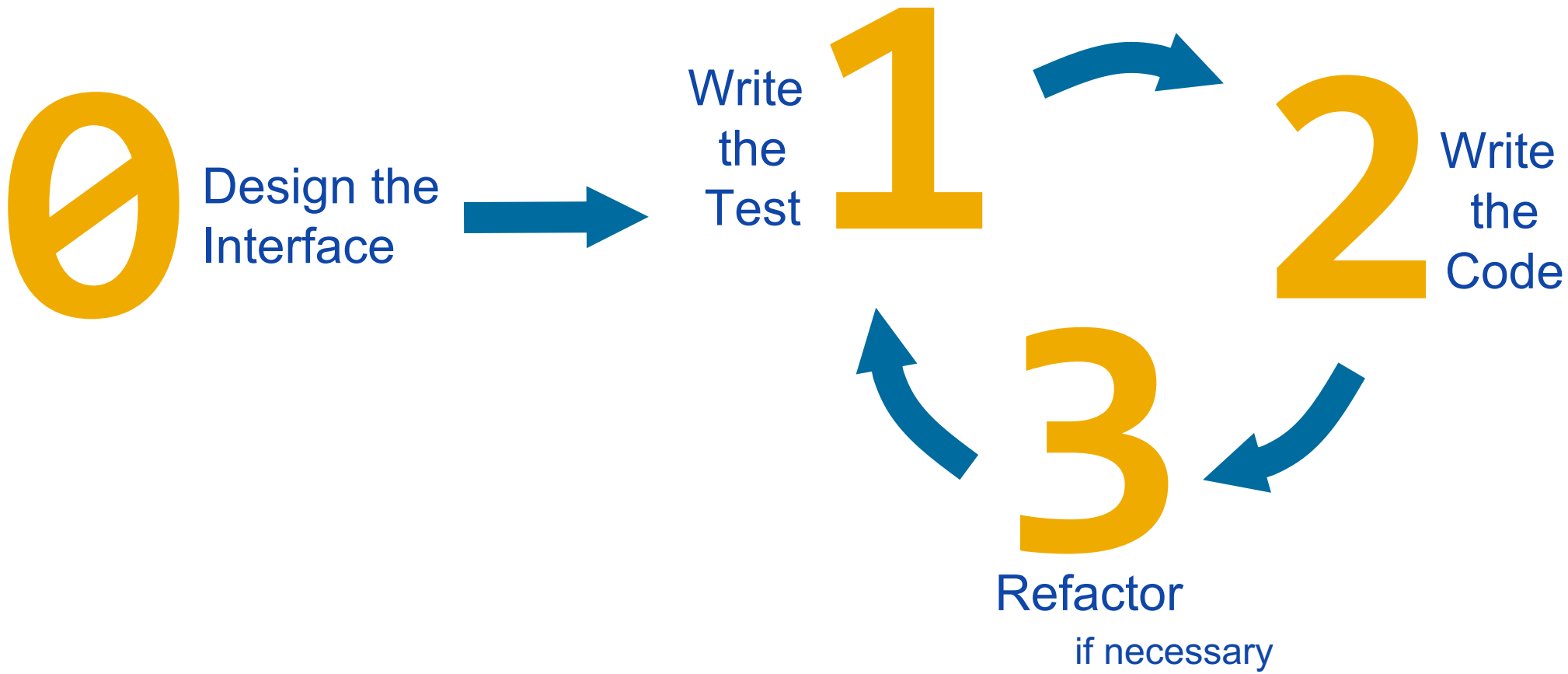
- The initial test should fail (because its requirements aren't implemented yet)
- Write code that will implement what is required
- Make sure the code passes the test
- Refactor

Benefits of TDD

- Helps you figure out the interface to your code
  - Improves the design
- Helps you stay focused on requirements rather than implementation details
- Encourages good test coverage and helps prevent future mistakes
- Creates living documentation for your code
  - The fine-grained tests show the expected behavior, which is usually clearer than documentation

Let’s be realistic!

- In an enterprise project, interface behavior is usually defined first by architects / designers.



JUnit Environment

JUnit

- An open source library for testing.
- Mainly suitable for unit testing.

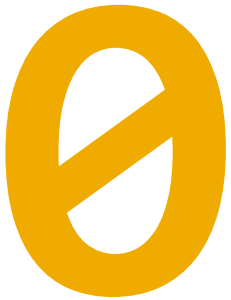
https://junit.org/junit5/

Testing with JUnit

Learn by Example

- Let’s apply TDD on a simple problem:
- Write a LinkedList data type that knows how to store, remove, and return its nodes, which are integers. Start with an interface called LinkedList, which you will later implement with the class DefaultLinkedList.

```
package my.tdd.demo;

public interface LinkedList {
    void append(int aNum);
    void remove(int index);
    int get(int index);
}
```



Design the Interface

Step 1 - Create the test class

```
package my.tdd.demo;
import static org.junit.Assert.*;
import org.junit.Test;
public class DefaultLinkedListTest {
   @Test
   public void testAppend() {
      fail ("Not yet implemented");
   }
   @Test
   public void testRemove() {
      fail ("Not yet implemented");
   }
   @Test
   public void testGet() {
      fail ("Not yet implemented");
   }
}
```

Write the Test

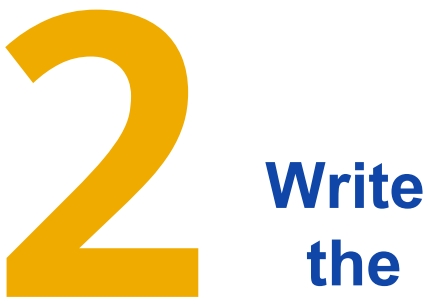
Step 1 - Implement the testAppend() method

- As an example, we'll implement the first test method
  - Note that we need to know the size of the list to verify that an integer was actually added
  - We'll refactor the interface in Step 3

```
@Test
public void testAppend() {
    // Scenario: add an integer to a linked list and see if its size changes to 1.
    LinkedList list = new DefaultLinkedList();
    assertEquals(0, list.size());
    list.append(42);
    assertEquals(1, list.size());
}
```

Step 2 - Write the Code: DefaultLinkedList

```
package my.tdd.demo;
public class DefaultLinkedList implements LinkedList {

   private int size;

   public DefaultLinkedList(){
      // TODO Auto-generated constructor stub
   }

   @Override
   public void append(int aNum) {
      // TODO Auto-generated method stub
   }

   public int size() {
      // TODO Auto-generated method stub
      return 0;
   }

   ...
   // not shown: remove() and get()
}
```



Step 2 - Write the Code: DefaultLinkedList - 2

```
package my.tdd.demo;
public class DefaultLinkedList implements LinkedList {
   private class Node {
      public Integer num;
      public Node next;
      public Node(Integer n) {
         this.num = n;
         this.next = null;
      }
   }
   private Node head;
   private int size;
   public DefaultLinkedList(){
      this.head = new Node(null);
      this.size = 0;
   }
   @Override
   public void append(int aNum) {
      // TODO Auto-generated method stub
   }
   public int size() {
      // TODO Auto-generated method stub
      return 0;
   }
   ...
```

- The implementation needs something to link together
  - We'll write a private class called Node.

Step 2 - Write the Code: DefaultLinkedList - 3

```
package my.tdd.demo;
public class DefaultLinkedList implements LinkedList {
   private class Node { ...
   private Node head;
   private int size;
   ...
   @Override
   public void append(int aNum) {
      // TODO Auto-generated method stub
   }
   public int size() {
      return this.size;
   }

   private Node tail() {
      Node tailNode = this.head;

      while(tailNode.next != null) {
         tailNode = tailNode.next;
      }
      return tailNode;
   }
   ...
}
```

```
private class Node {
   public Integer num;
   public Node next;
   public Node(Integer n) {
      this.num = n;
      this.next = null;
   }
}
```

- The size method just needs to return the number of integers in the list
- The append(int) method needs to know the last node in the list.
  - Write the tail() method to return that.

Step 2 - Write the Code: DefaultLinkedList - 4

```
package my.tdd.demo;

public class DefaultLinkedList implements LinkedList {

   private class Node { ...

   private Node head;
   private int size;

   public DefaultLinkedList(){
      this.head = new Node(null);
      this.size = 0;
   }

   @Override
   public void append(int aNum) {
      Node newNode = new Node(aNum);
      Node tailNode = tail();
      tailNode.next = newNode;
   }

   public int size() {
      return this.size;
   }

   private Node tail() { ...

   ...
}
```

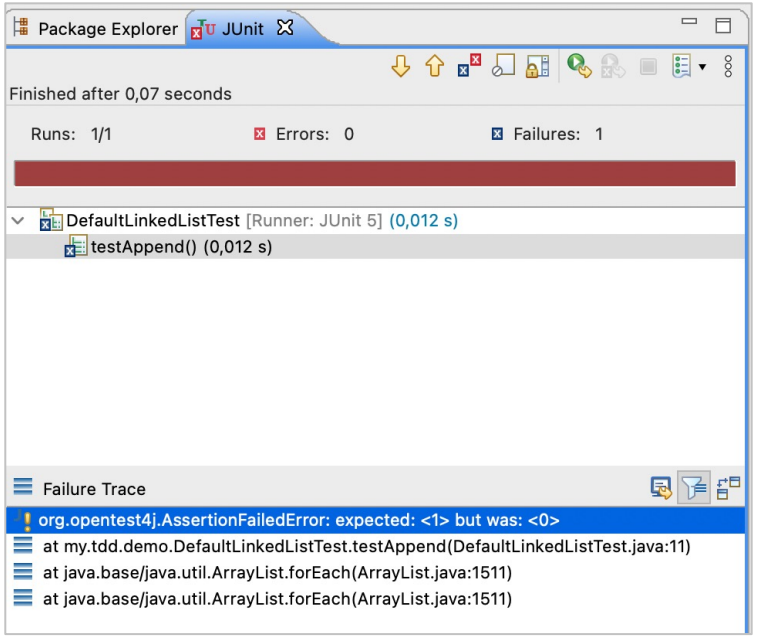
```
private class Node {
   public Integer num;
   public Node next;
   public Node(Integer n) {
      this.num = n;
      this.next = null;
   }
}
```

- We can now write the append() method

Step 2 - Write the Code: Test DefaultLinkedList

- Keep coding until the code passes the test.
  - So, let's run the test

Hey!
The size seems to be wrong...



Step 2 - Write the Code: DefaultLinkedList - 5

```
package my.tdd.demo;

public class DefaultLinkedList implements LinkedList {

   private class Node { ...

   private Node head;
   private int size;

   public DefaultLinkedList(){
      this.head = new Node(null);
      this.size = 0;
   }

   @Override
   public void append(int aNum) {
      Node newNode = new Node(aNum);
      Node tailNode = tail();
      tailNode.next = newNode;
      this.size++;

   }

   public int size() {
      return this.size;
   }

   private Node tail() { ...
}
```

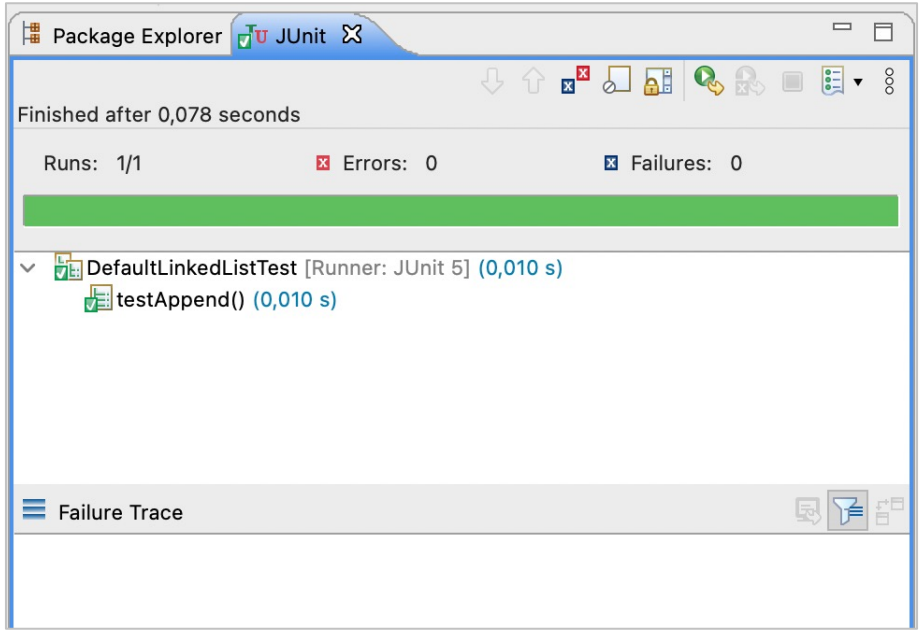
```
private class Node {
   public Integer num;
   public Node next;
   public Node(Integer n) {
      this.num = n;
      this.next = null;
   }
}
```

- Oh, that's right... we forgot to update the list's size after adding a node!

Step 2 - Write the Code: Test DefaultLinkedList - 2

- Run the test again...

Great!



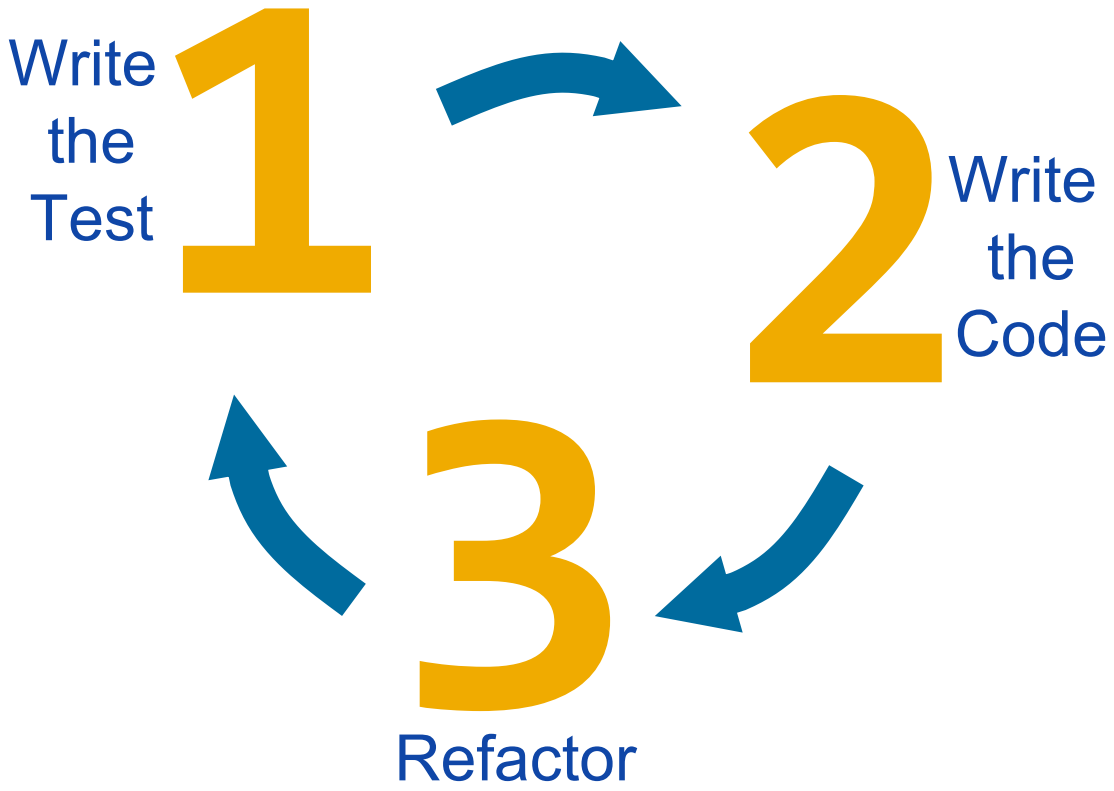
Step 3 - Refactor the LinkedList interface

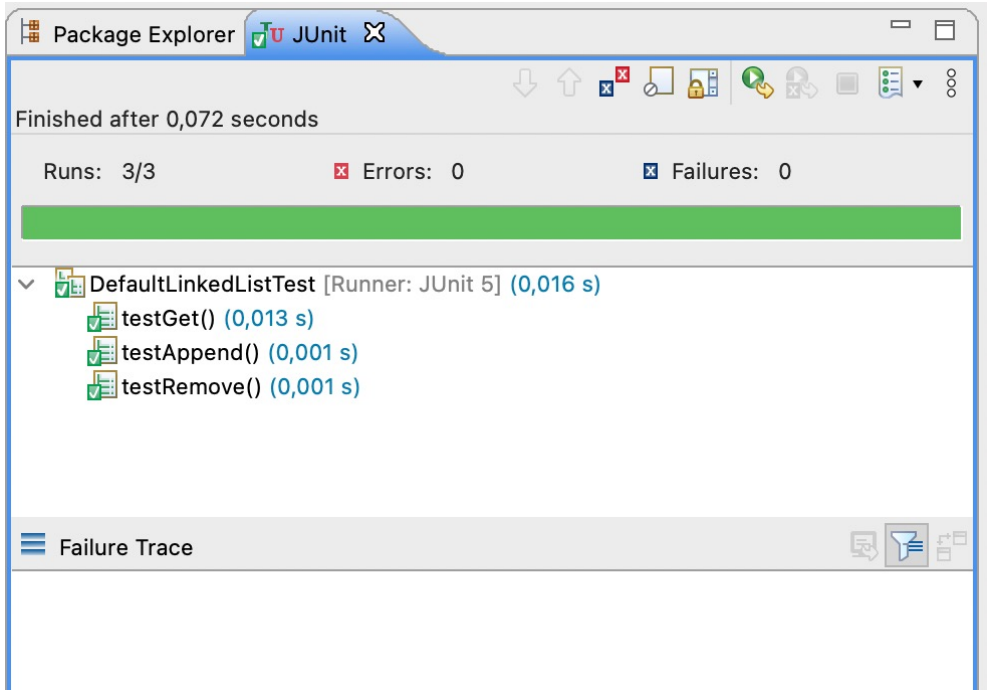
- Finally, refactor the interface to account for the public method we added: *int size()*

```
package my.tdd.demo;

public interface LinkedList {
    void append(int aNum);
    void remove(int index);
    int get(int index);
    int size();
}
```

LinkedList – Continue development ...

- The cycle continues for the next chunks of functionality
  - In this case, that would be the remove() and get() methods...





Thank you.