# static import

Monday, June 1, 2020 5:20 PM

```java
package in.conceptarchitect.tests;

//import all static methods from Assert class
//this way all static method of the calss can be invoked without using Class reference
import static org.junit.Assert.*;

import org.junit.Test;

public class LinkedListTests {

    @Test
    public void test() {
        fail("Not yet implemented"); //actually imported method Assert.fail()
    }

}
```

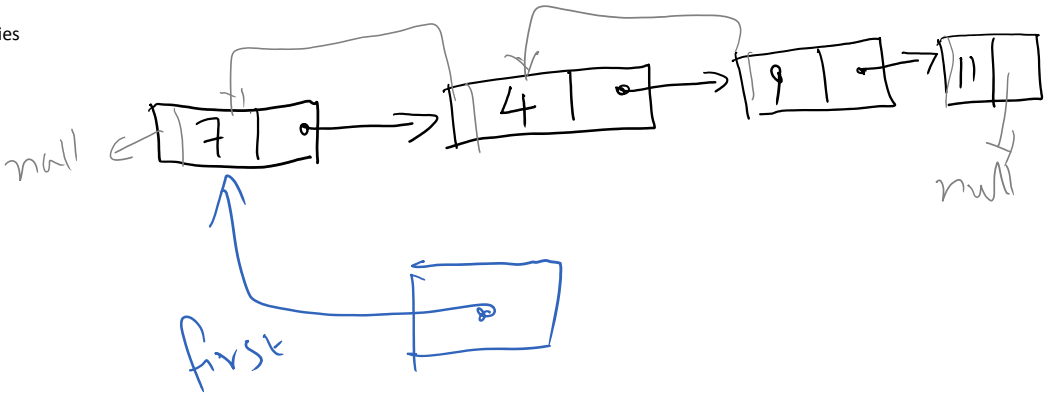
Allows all static method of a class to be imported as a global method

These methods don't require class name to call them

# Assignment01 LinkedList

Thursday, May 21, 2020 2:58 PM

- Create a class to represent a Linked List
- A Linked List should support following operations
  - **add( int value)** //Adds to end of the List
  - **get(int pos)** //get a value from a given position
  - **set(int pos)** //set a valjue to a given position
  - **size()** //returns the size of the list
  - **remove(int pos)** //remove the value from a given position
- Create the necessary classes
- Write a **main function** to test its functionalities




## List Access

- A list should allow random access (logically)
  - LinkedList is however physically sequential
  - To access it random we need to internally travel sequential.
- A List supports two common access requirements

### 1. Direct Access a.k.a Random Access

- User may need to access value at index 27, 49,112,4,18,37
  - This requirement is likely
  - we have **get(int pos)** to handle this requirement

### 2. Access all items one by one

- in a sequence
- for-each item
- This appears to be a very popular use case

We can't do much to improve performance of direct access

**Optimization**

- may be improving performance for specific use case
- May not be for all use case

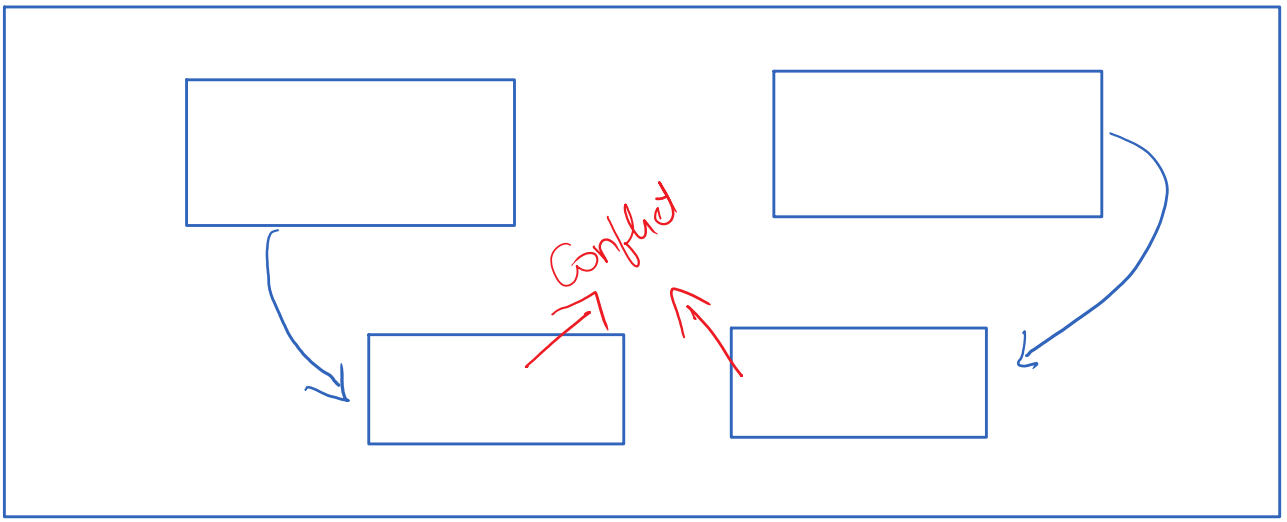
But we can improve the performance of sequential access (Popular Use case)

- ***I we have accessed an item at position 'n' the next request is going to be more likely for position 'n+1'***

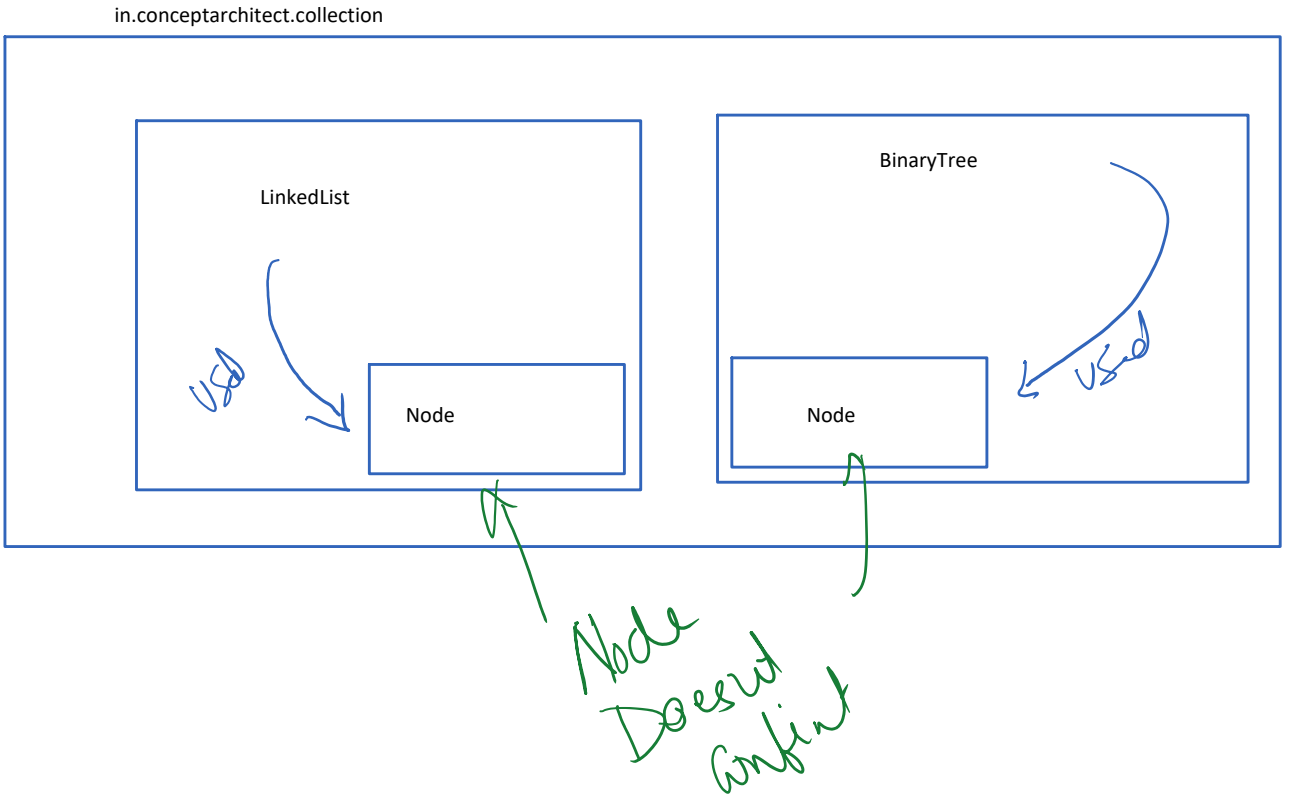
**Can we maintain a pointer on the last accessed position.**

# Package vs Class Boundry

Friday, May 22, 2020 3:57 PM




## A class to can act as a Package to separate class name visibility




## When should I user inner class

- The outer class uses the objects of inner class **exclusively**
- The inner class object is not directly utilized by anyone else
- The only purpose of inner class is to support the outer class

## Not every child component should be inner class

- A car contains tyres
- But a Tyre has independent existence and manufacturer
- We will not define Tyre class as inner class to Car

# Packaging best practice guidelines

Monday, June 1, 2020 10:50 AM

## Do's

- Make sure, your reusable components that can be productive to more than one applications, should be int its own
  - Package
  - Jar
- A Package is **Not designed to hold a single class**, but it is designed to hold a similar or related set of classes
  - Good Examples
    - collection —> to hold collection classes related to data structure
    - sql —> classes related to database access
    - net —> Network related classes
    - swt —> database related classes
  - Bad Examples
    - util —> to hold unrelated utilities such as Date, StringBuilder, Scanner, LinkedList
      - **java.util is an example of bad example**

- A Sub package may contain more specific elements from the super package
  - GoodExample
    - net.http —> classes related to http protocol which is a type of network protocol
    - jface.text —> text related elements in jface
- Top level package should be an identity space
    - **java.**sql
    - **java.**awt
    - **org.eclipse.**swt
    - **org.eclipse.**jface
    - **org.eclipse.**jface.text
    - **in.conceptarchitect.**collection
    - **in.conceptarchitect.**utils
    - **in.conceptarchitect.**taskmanager <—objects related to task manager application
    - **in.conceptarchitect.**taskmanager.ui <— ui layer of task manager application
    - **in.conceptarchitect.**taskmanager.repository <-- data access layer of taskmanager application

- Same rule applies to Jar also
  - However a jar can have multiple Packages
  - **org.eclipse.jface.jar** may contain all **jface** packages and subpackages
  - **Remember:** jar is the smallest unit of deployment
- **internal and inner classes**
  - You should limit the visibility of those classes that are for **internal usage only** and which the client shouldn't access.
  - To limit the visibility we have three choices
    1. use package level class (don't make it public)
       - This is an elementry security
       - Client can create package with same name and can still access it
    2. Make private inner classes
       - No one within the package can access it
       - Client's can't access
       - Not always possible
    3. Use Java9 Module system <— discussed later

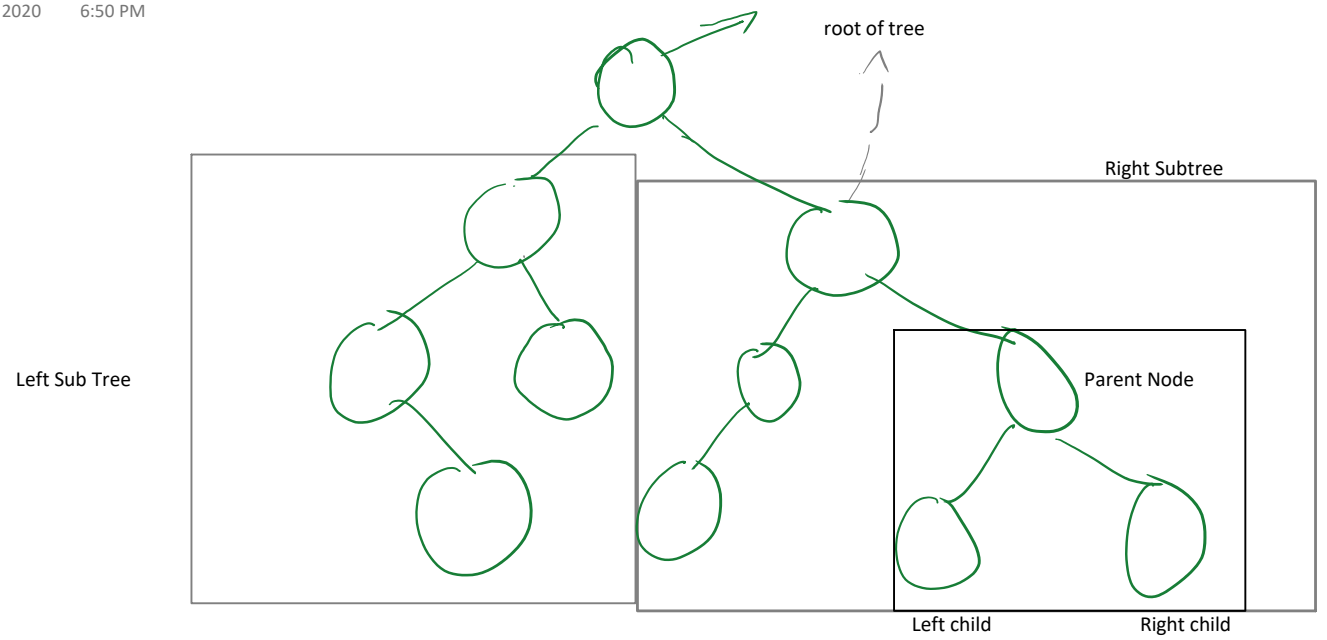
## Don'ts

- Don't keep **main()** in your component class
- **Always remember main() should be in its own class in the client jar**
- Don't create single level package
    - It must have a brand identity
        - You may use a fictious brand such as **com.yourname**
- Don't create meaningless package
- A good structure for simple practice exercise could be
    - **jar: com.myname.collection**
        - **package: com.myname.collection**
            - **class LinkedList**
                - **class Node**

    - **client:**
        - **option1**
            - **com.myname.testapp.linkedlist**
                - **package: com.myname.testapp.linkedlist**
                    - **class: Program (or Test or App or Client)**
                        - **method: main()**
        - **option2 (relaxation)**
            - **jar: testapp01.linkedlist <— this makes seeing the package explorer easy**
                - **This is just a test application which is a throaway later**

# BinaryTree of int

Friday, May 22, 2020 6:50 PM



# BinaryTree Create Rule

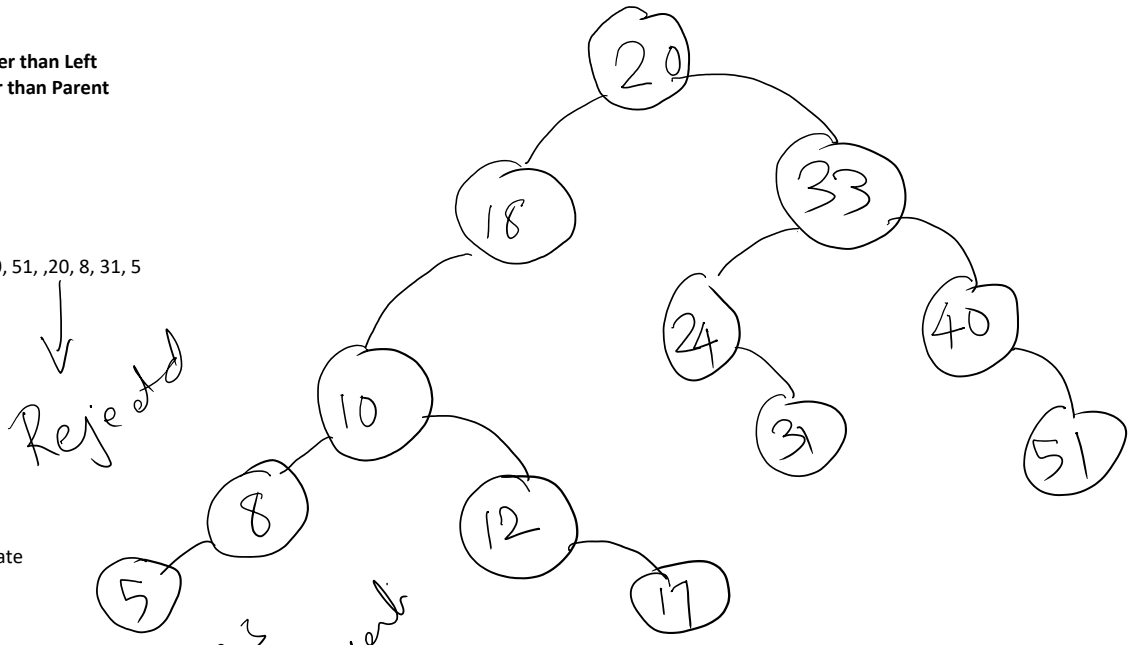
Friday, May 22, 2020

**Tree Rule: L<P<R**

- **Parent should be greater than Left**
- **Right should be greater than Parent**

To Add following Numbers

20 , 18, 10, 12, 17, 33, 24, 40, 51, ,20, 8, 31, 5




- If you want to retain duplicate you can change the tree forumula to one of the two given below
  - L<=P < R
  - or L<P<=R
- But Not
  - L<=P<=R

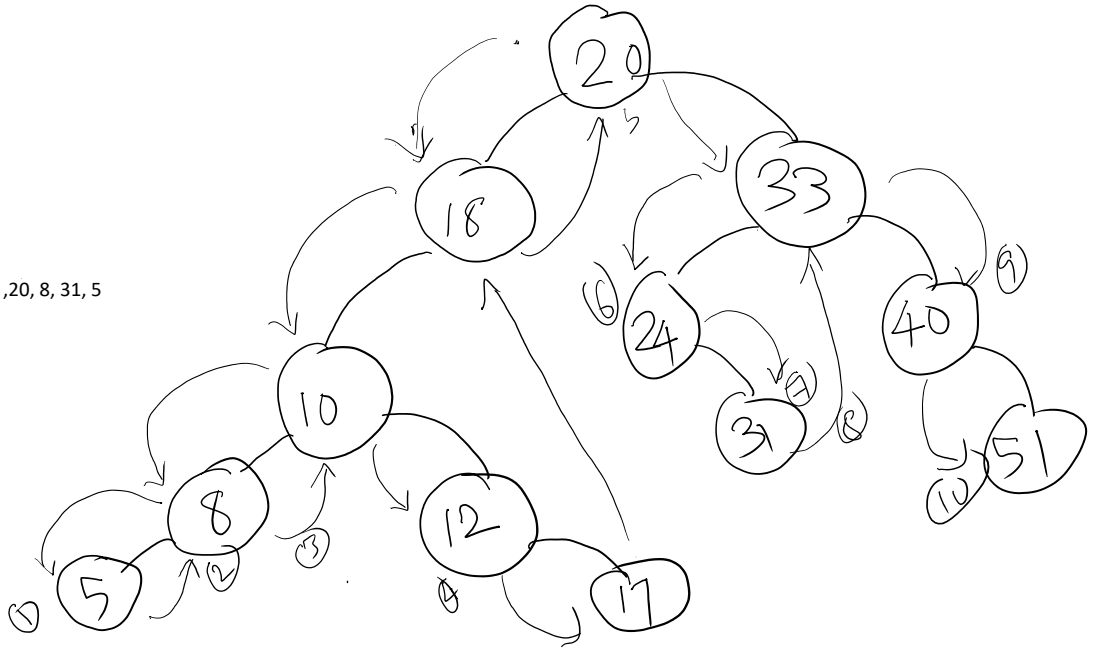

```
Node  insert ( Node root, int value){

        if(root==null){
                root=new Node(value);

        } else if(value< root.value)
                root.left=insert(root.left,value);
        else if(vlaue> root.value)
                root.right=insert(root.right,value);
        return root;
}


}
```

# BinaryTreeAccess Rule -- Inorder

Friday, May 22, 2020

**Inorder: L-->P-->R**

- **Visit Left (subtree)**
- **Visit Parent**
- **Visit Right(subtree)**




To Add following Numbers

20 , 18, 10, 12, 17, 33, 24, 40, 51, ,20, 8, 31, 5

Inorder

5
8
10
12
17
18
20
24
31
33
40
51

```
Node inorder ( Node root){

        if(root==null){
                return;

        }
        inorder(root.left); //L
        print(root.value);  //P
        inorder(root.right); //R


}
```

## Preorder

**P --> L --> R**

## Preorder

**L --> R --> P**

# Assignment 02

Friday, May 22, 2020 7:10 PM

- create class BinaryTree to store integers
- Implement operations
    - Insert
    - Inorder
    - Preorder
    - Postorder

# Same Logic Different Data

Monday, June 1, 2020 11:15 AM

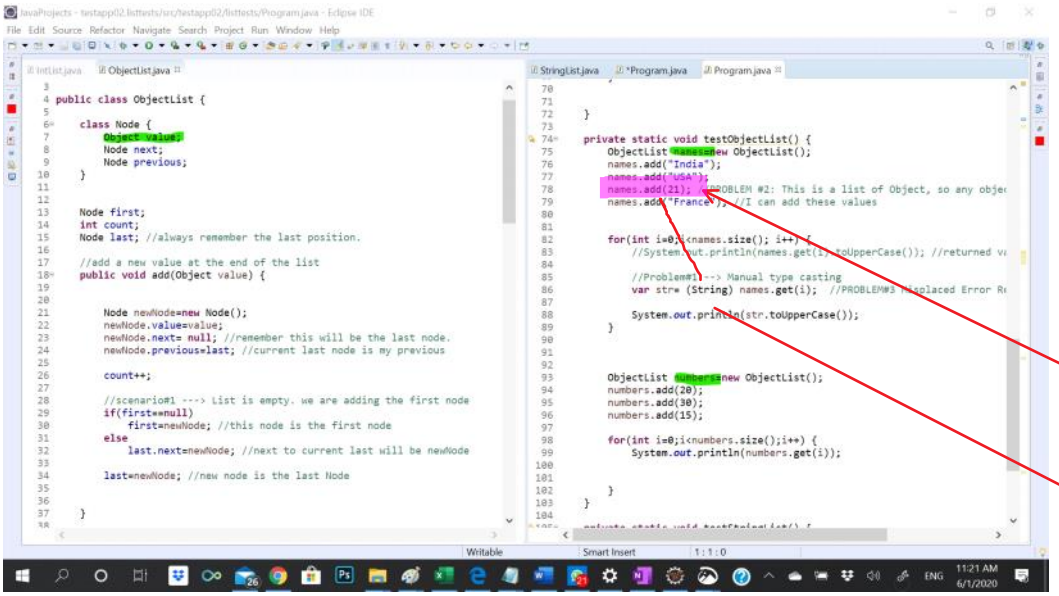

```
public class IntList {

    class Node {
        int value;
        Node next;
        Node previous;
    }


    Node first;
    int count;
    Node last; //always remember the last position.

    //add a new value at the end of the list
    public void add(int value) {


        Node newNode=new Node();
        newNode.value=value;
        newNode.next= null; //remember this will be t
        newNode.previous=last; //current last node is

        count++;

        //scenario#1 ---> List is empty. we are addin
        if(first==null)
            first=newNode; //this node is the first n
        else
            last.next=newNode; //next to current last

        last=newNode; //new node is the last Node


    }
```

```
public class StringList {

    class Node {
        String value;
        Node next;
        Node previous;
    }


    Node first;
    int count;
    Node last; //always remember the last position.

    //add a new value at the end of the list
    public void add(String value) {


        Node newNode=new Node();
        newNode.value=value;
        newNode.next= null; //remember this will be t
        newNode.previous=last; //current last node is

        count++;

        //scenario#1 ---> List is empty. we are addin
        if(first==null)
            first=newNode; //this node is the first n
        else
            last.next=newNode; //next to current last

        last=newNode; //new node is the last Node


    }
```

- Because the LinkedList algorithm doesn't know or care to know what is the data type
- it doesn't try to use any internal functionality or property of the data
- It is simply storing the data at the end without caring the exact value or meaning of data.
- If your algorithm needs to call special methods from the data, it can't be used as a generic alorithm easily.

# Object List

Monday, June 1, 2020 11:22 AM




## Good

- Same LinkedList class can allow you to create linkedlist to hold different type of data
  - String
  - Date
  - Task
- You don't have to create different classes, just different objects

## Bad

- class doesn't know what kind of object you want to store in linked list. so it allows you to store even number in a list of Strings
- returns from object method will be an object and should be typecasted before used. You don't get intellisense unless you typecase
- if you stored wrong value, the typecasting will faile
-

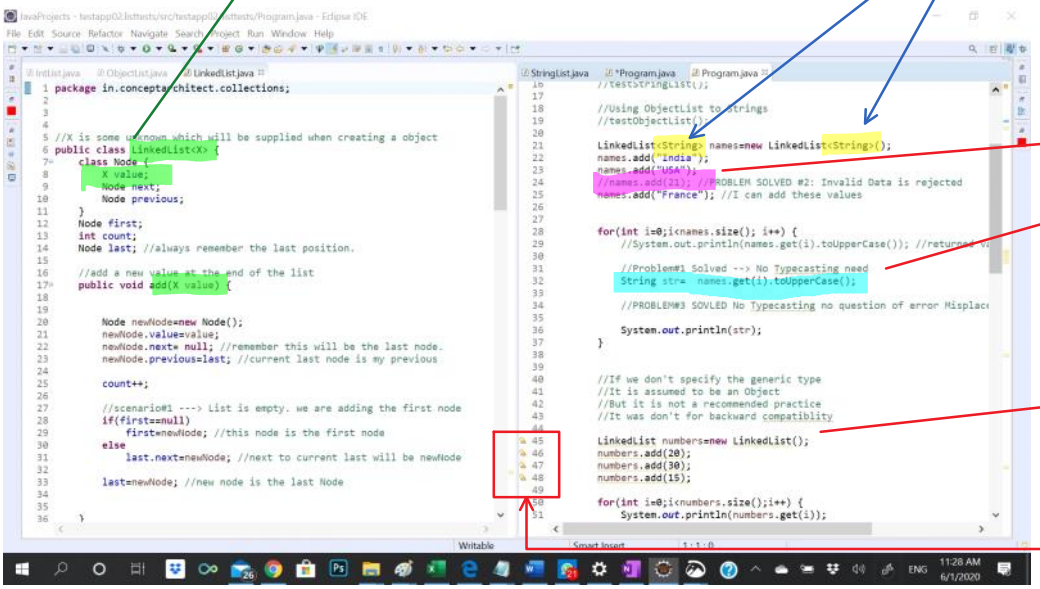
# Generics

Monday, June 1, 2020 11:29 AM

1. Class is Created in terms of unknown like an Alzebric unit
   a. It can be any valid identifier that you may create
   b. Generally java uses **E** for element

2. The data type must be specified while creating the object

After java 6, it is ok to mention type only on the left side and not on right side.




**The benefits**

1. **Detects and complains for error early**
2. **Use object without typecasting**

**Java Implementation**

- **Java allows to create object without specifiying type.** This is for backward compatibility and is Not Recommended
- When you don't specify the type it falls back to Object type
- **This is not Recommended and java complains by giving warning**

# Generic is internally Object

Monday, June 1, 2020 12:01 PM

- When Java created Generics, it was a language level feture and **Not byte code feature.**
  - **JVM was not expected to understand generic**
- Java internally converted a Generic type **X** to an **Object** type
  - It internally checked if you are breaking any rule by inserting wrong value type
  - Intellisence is a combined feature of compiler and the IDE.
- Once a java generic is compiled, it becomes Object.

  **LinkedList<String> list=new LinkedList<String>(); // This code is essentially same as**
  **LinkedList<Object> list=new LinkedList<Object>(); // This code is essentially same as**
    - with compiler checking if you are trying to insert anything other than String.

- That is why when you don't specify Generic during object creation it becomes Object

  **LinkedList list=new LinkedList(); // This code is essentially same as**

  **LinkedList<Object> list=new LinkedList<Object>(); // This code is essentially same as**
    - With compiler making no checks.

## Problem — You can't create LInkedList of int

**LinkedList<int> list=new LinkedList<int>();**

- Why?
  - because in java **int is not a primitive type and not an Object type**
  - Java Generic convert to Object and int can't be object.

## Solution — This is not a big problem in the first place.

- We can use following syntax

**LinkedList<Integer> list=new LinkedList<Integer>();**

- Integer is a wrapper **class** around int
- **Integer** is a **class type** that extends **Object**
- Java provides autoboxing and auto unboxing between Integer and int

//auto boxing

Integer i= 49; //—> it is same as **Integer i=new Integer(49) —> This is autoboxing**

int j= I ; //—> It is same as **int j= i.intValue(); —> Auto boxing**

## How to use LinkedList<int>

1. create a **LinkedList<Integer>** not ~~**LinkedList<int>**~~
2. Add int value normally --> autoboxing will convert int to integer
3. Access int value normally —> autounboxing will convert Integer to int

# Accessing All Items of a List

Tuesday, June 2, 2020 3:28 PM

## Manual way

- Remember the last access index and node
- if the request is for next item, it will be fast,
- else go sequential.

## Implementing Iterator Pattern

- Sequential access is such an important use case that it is a **Design Pattern called Iterator.**
- Iterator defines an infrastructure so that each Item can be accessed in a given sequence one after another
- Java defines an Iterator interface that you should be implementing

### Java Iterator Interface

- next()
  - return a value and moves to next item
- hasNext()
  - reurns a if there is a next item

- Java Iterators are generally implemented using an inner class

- A class that has an iterator is Iterable.

```
pubilc interface Iterable<T>{
        Iterator<T> iterator()
}
public interface Iterator<T>{

        T next();   //returns the next item

        boolean hasNext();  //tells if there
        is a next item avaialble

}
```

```
class LinkedList<T> implements Iterable<T> {

        public Iterator<T> iterator(){
                return new MyIterator();
        }

        class MyIterator implements Iterator<T>{
                public T next(){

                }

                public boolean hasNext(){

                }

        }
}
```

### How do I Use Iterator

- There are two way to use Iterator
  1. Traditional Object Oriented Way
  2. using for loop

### 1. Using Object Oriented Approach

```
@Test
public void accessIterableValues(){

        Iterator<Integer> it= list.iterator();
        while(it.hasNext()){
                int value= it.next(); //instead of list.get()
        }
}
```

# Print and Main Based Test

Monday, June 1, 2020 12:56 PM

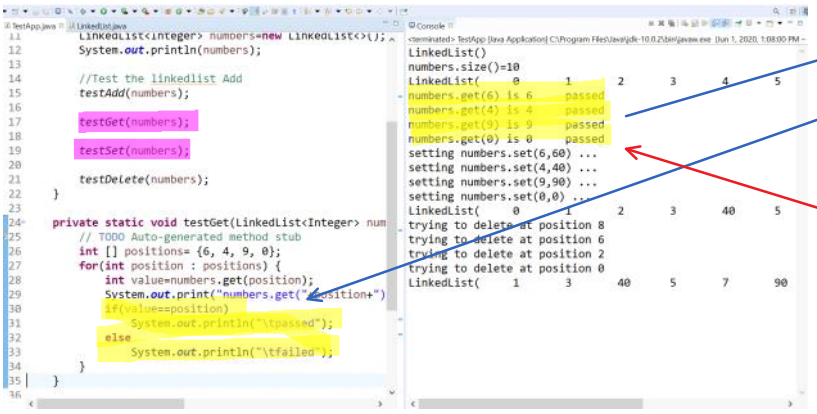
## main() function wasn't designed to test your code. It was to run a tested code

- print() is for output and the output is for **Humans**
- with a print() outptu you must look and verify if the result is expected or not
  - system can't decide for your
  - This is a manual testing process not automated testing process.
- main() is not for testing, its to run one core activity
  - Test should test different part of a system


### Problems

1. How do we know where first test ends and second begins?
   a. Which test prints this line — testAdd or testGet?
2. How do I know if the printed value is the expected value?
   a. Even wrong value printed will still be an output.
   b. Are we sure we expected this output?

## One Result Can Influence Other Result

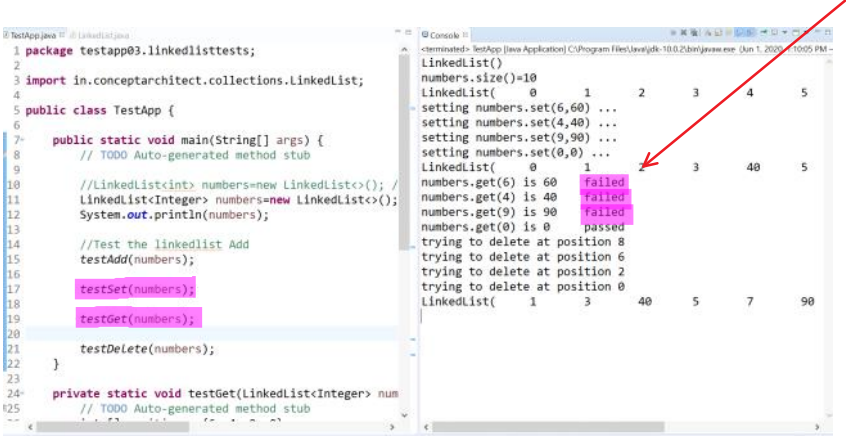


Test Results can be interpreted better

if there is a good logic written

### Problem

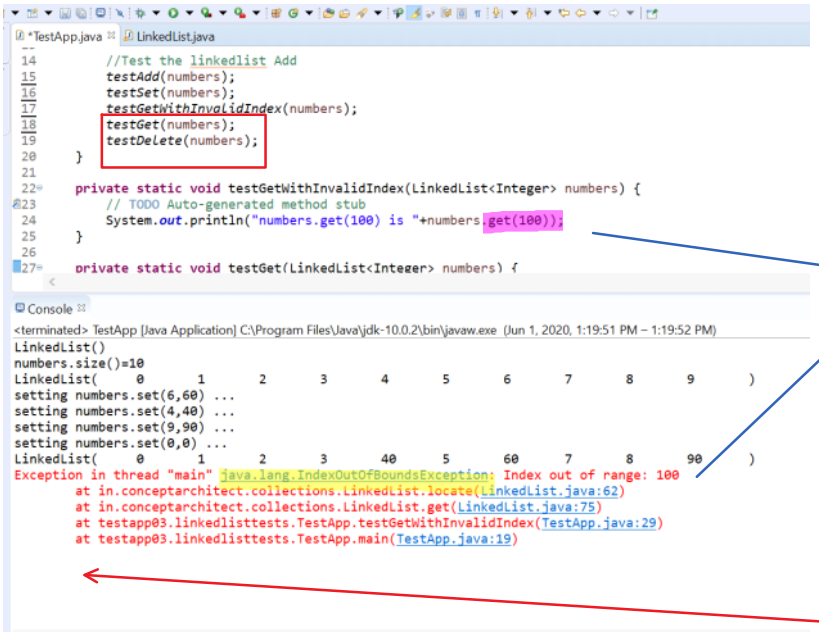
3. Test Result Changes when we change the order of the test.
   a. Are we sure getFunction is working currectly?
   b. which of the two gives test result more accurate?
   c. How to we decide the correct sequence?
   d. Will user of my code use the code in same sequence?
      i. Do they need to call get becore calling set?
   e. Result is changing based on call sequence



### Problem 5

- Are we sure we have tested all scenario?
- Is my application Working correctly with invalid index?

- Is my application working correctly with invalid index?

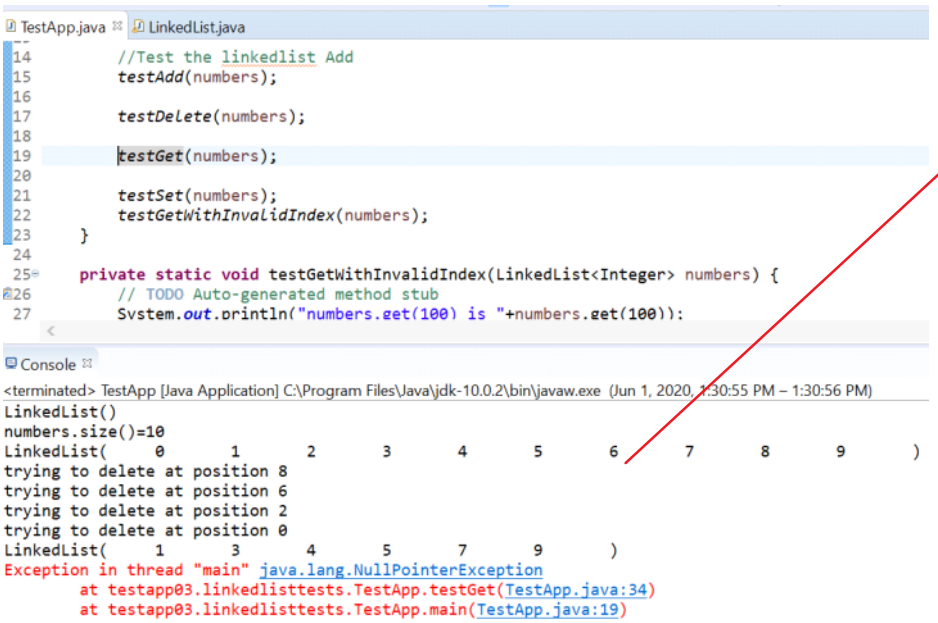



## Problem 5.1

- Is the result a proof of success or a proof failure?
  - Does this exception mean success or fail?
- For a invalid index (100) my code is expected to throw **IndexOutOfBoundsException**
  - Since we are getting what we are expecting the LinkedList Code is working correctly (as per expectation)
  - But Human eyes see
    - Red as Trouble
  - Developers eyes see
    - Exception as Red as Trouble

## Problem 6

- What about the remaining tests — testGet() and testDelete()?
- You see they haven't executed.
  - Exception breaks the program




## Most Important Problem

- Is this just a sequencing problem or a real error?
- Error exists in testAdd(), testDelete() or testGet()
- Is there a bug in LinkedList add(), get(), delete()

## Summary

1. print is for human eyes.
   a. A causal glance may not tell you if result is expected or not
   b. Wrong result is also printed the same way as right result
   c. Makes testing manual, system can't tell it worked or failed
   d. test boundries are not clear
2. test results influence each other
   a. reording the sequence may cause wrong answers even if there is no bug in the code
3. Sad path testing (Exceptions) may look like a failure even when they are success
4. Exception breaks the exuection of application so remaining test may not execute
5. When a bug comes it may be due to
   a. calling all functions together
   b. due to a function which had bug but was not discovered earlier
6. Since we are calling several functions we are not sure who the real culprit is.

# Unit Testing Framework

Monday, June 1, 2020 1:39 PM

- Modern age testing tools
- Special framework to make testing easy

## Qualities of a Good Testing Framework

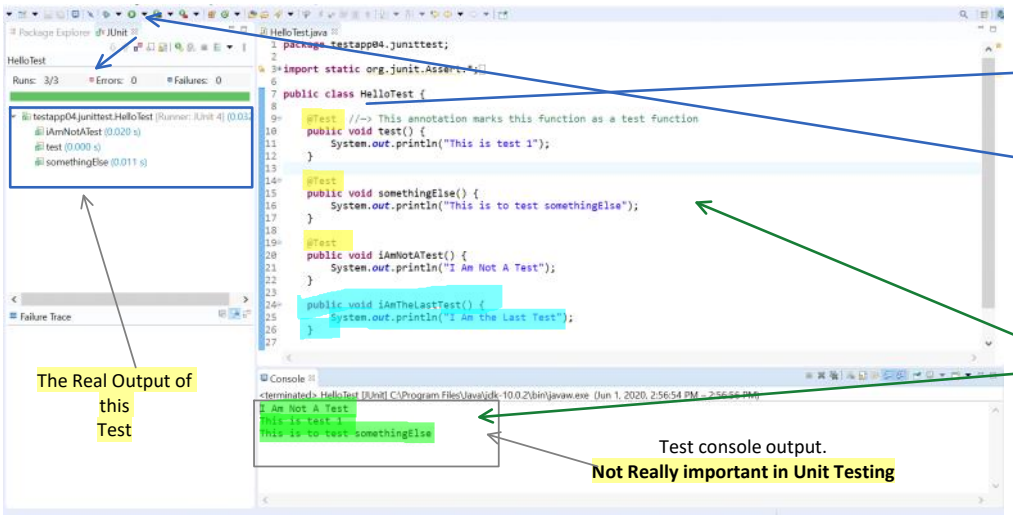
1. Automatic
    a. Can detect if the test is giving correct result or not
        i. Not based on main() and print()
2. Atomic
    a. Each test is expected to test a very small atomic unit of the code and ensuring this piece works
3. Isolated
    a. Tests should not influence each other. They all should work independently
        i. easy to find out the real problem
4. Sad Path
    a. Should also successfully test the SAD path

## Junit

- Junit is a unit testing framework for Java language
- It the first unit testing framework in any programming language.
- It influeced the design of testing frameworks across all programming languages.

# Junit Test Design

Monday, June 1, 2020 2:57 PM




The Real Output of this Test

Test console output.
**Not Really important in Unit Testing**

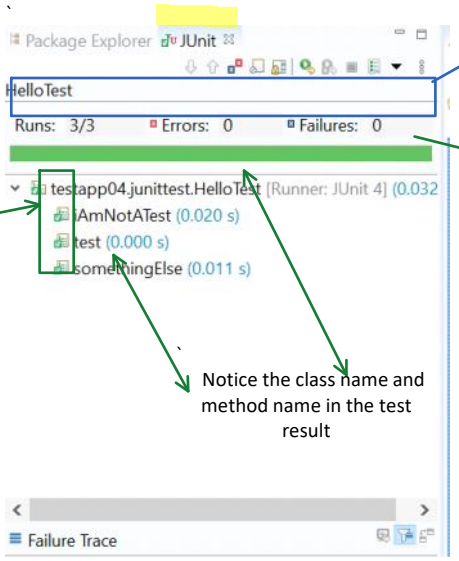
1. Marks our class and this method as A Test Method.
2. A Test Method is executed by a Test Framework
3. Methods that are marked as **@Test** are executed, others ignored

**IMPORTANT!**

- **Test doesn't have order.**
- **Order Is not important**
- **Remember: Each Test is Isolated**

# Test Explorer

Monday, June 1, 2020 3:09 PM




Test Summary

- 3 our of 3 Test Executed
- Total Errors 0
- Total Failures 0

Tick against individual tests

- gree indicates success

**Green Bar**

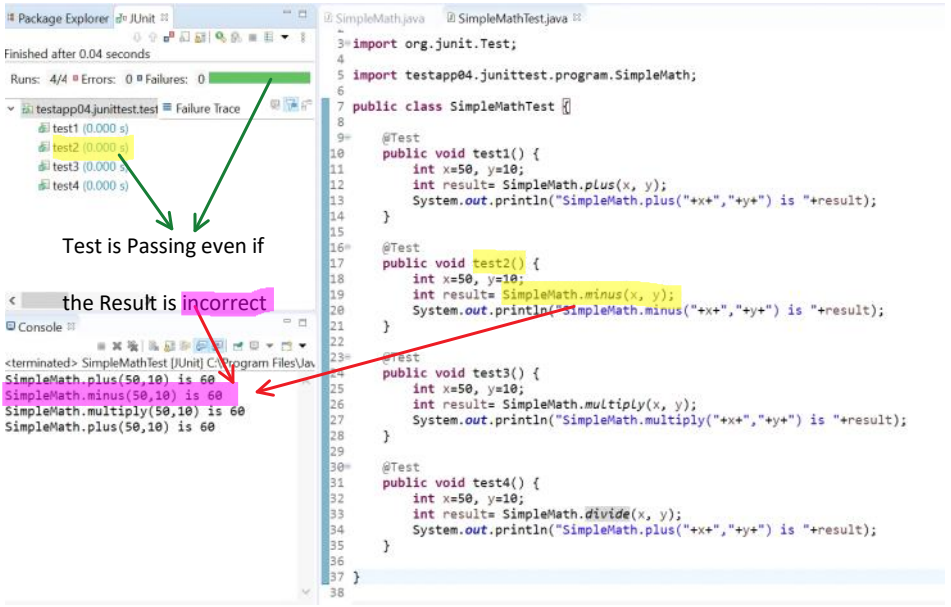
- Junit uses green color to mark success
- You get this green bar only if All your tests are success.
- If any test fails, the bar will be dark red (brown)

Why has all test passed?

Notice the class name and method name in the test result

# What is a Test Pass or fail?

Monday, June 1, 2020 3:29 PM



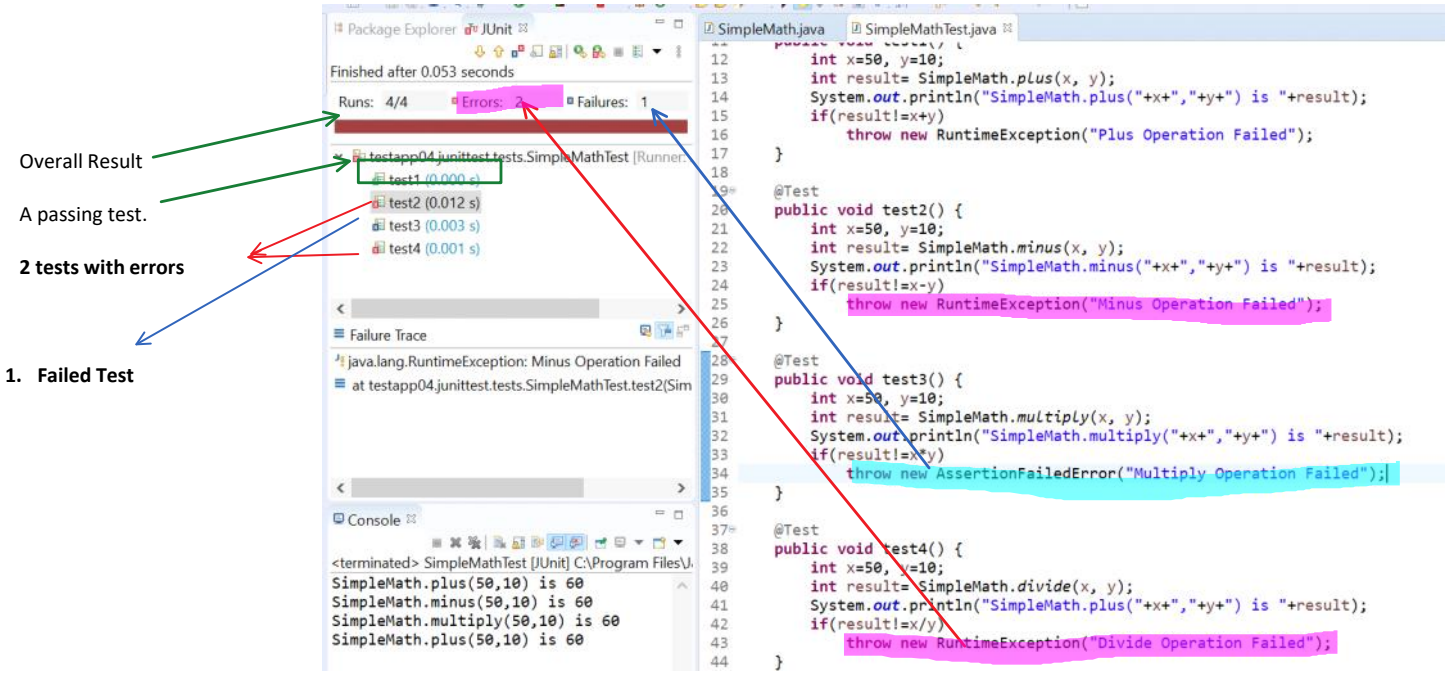

## Why does the test pass?

- Junit doesn't know what is the expected output
- If wrong result printed is an output
- We follow a simple rule

**No news is a good news. So unless there is something Exception wrong, it is a success.**

# Test with Errors

Monday, June 1, 2020 3:36 PM

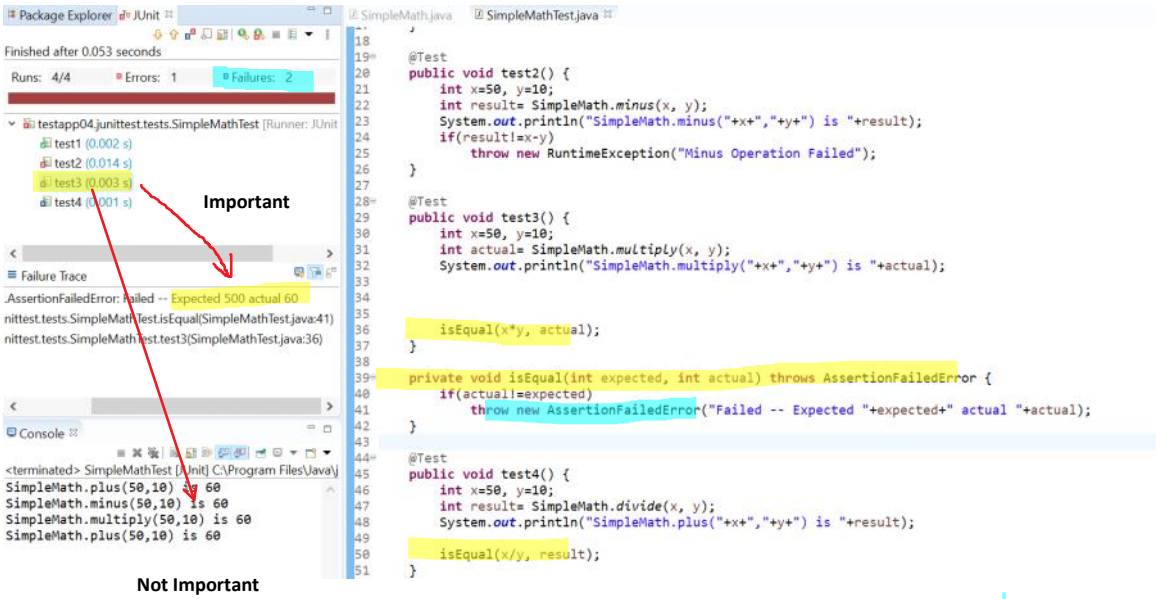



## What is the difference between an Error and A failure

- The purpose of a unit test is to indentify if the code is working as expected
  - expected working => function gives the expected result
- A function that gives unexpected result is a **failure**.
  - Function completes execution
  - It returns a result
  - The result is not what we expected.
  - Internally jUnit throws AssertionFailedException to indicate failure
- If a function fails to complete it is an error
  - If a function throws an exception while execution
  - Its execution is not complete
  - It has not produced a result to be considered success or failure
  - It is considered as an error.
  - Any exception other than AssertionFailedError make it an Error

# Error And Failure

Monday, June 1, 2020 3:55 PM




We can have Test Helper

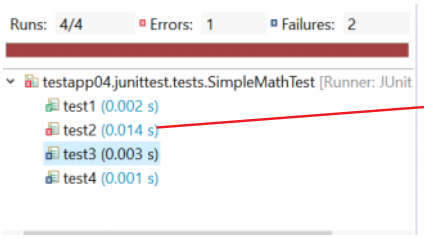
that throws

AssertionFailedEerror

in case the expected condition is not me

# Test Design Practices

Monday, June 1, 2020 4:01 PM

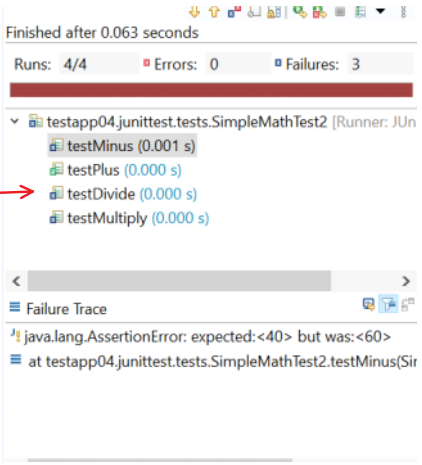



## What does test2 do?

- Which programming logic has error?
- How do I know looking at this test result alone?
- What does test2 represent here?

## What does this method do?

- Which one has failed?




Your Test Names should be meaningful

# Assertion Library

Monday, June 1, 2020 4:09 PM

```java
@Test
public void testMinus() {
    int x=50, y=10;
    int result= SimpleMath.minus(x, y);
    System.out.println("SimpleMath.minus("+x+","+y+") is "+result);
    Assert.assertEquals(x-y, result);
}

@Test
public void testMultiply() {
    int x=50, y=10;
    int actual= SimpleMath.multiply(x, y);
    System.out.println("SimpleMath.multiply("+x+","+y+") is "+actual);


    Assert.assertEquals(x*y, actual);  //isEqual(x*y, actual);
}

private void isEqual(int expected, int actual) throws AssertionFailedError {
    if(actual!=expected)
        throw new AssertionFailedError("Failed -- Expected "+expected+" actual "+actual);
}

@Test
public void testDivide() {
    int x=50, y=10;
    int result= SimpleMath.divide(x, y);
    System.out.println("SimpleMath.plus("+x+","+y+") is "+result);

    isEqual(x/y, result);
}
```

Both are conceptually same. **isEquals** is our own logic
Assert.assertEquals is a junit library that does the exact same job

jUnit has provided several such functions to Assert on your result
You get a failure when your result is not as per expectation. Common Assert includes

- assertEquals
- assertNotEquals
- assertTrue
- assertNotNull
- assertNull
- fail() <— absolute failure

# You may need to write multiple test for a single method

Monday, June 1, 2020 4:14 PM

**Example:**

- **Is divide working correctly if denominator is non-zero**
- **Is divide working correctly if denominator is zero**

## How do I name different tests related to same divide method?

- divideReturnsCorrectResultForNonZeroDenominator()
- divideThrowsExceptionForZeroDenominator()

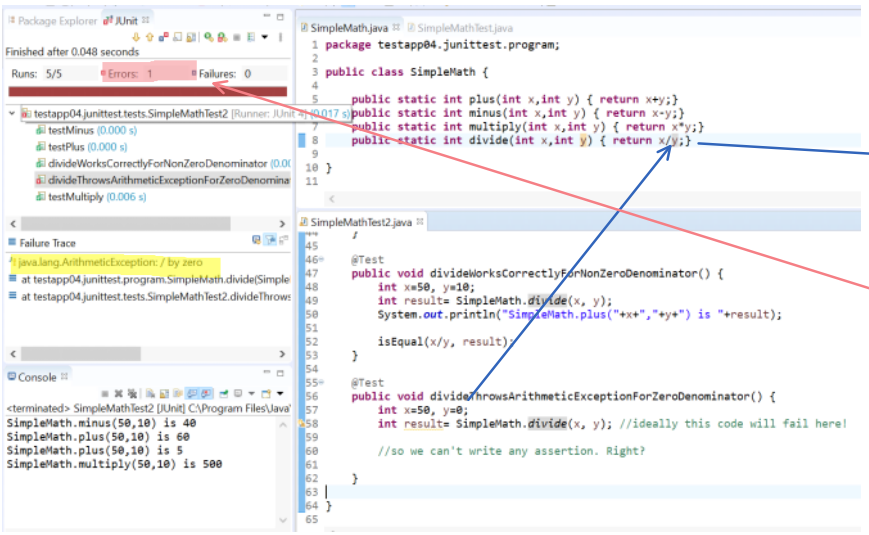
A test name should follow **DAMP** principle

**DAMP --> Descriptive and menaingful phrases**

- **Normally you method names should menaingful words.**
- **Your test metods should be longer and descripive phrases or sentenses not just words**

# Asserting For Exception

Monday, June 1, 2020 4:22 PM




If y == 0 it throws ArithmeticException

It is an expected behavior
So the test should pass

But since it is a error it is categorized as Error

## Important!

- Even if except is thrown, The tests continued to execute without any interruption.
- Failure of one test case doesn't effect the other.

## How to handle expected exception

### 1. User define approach

```
@Test
public  void divideByZeroShouldThrowArithmeticException(){

	try{
		SimpleMath.divide(7,0); //should throw ArithemeticException

		//If I reach here. It means exception is not throw  and
		//the test has failed
		fail("excpected exception ArithmeticException wasn't thrown";
	}catch(ArithmeticException ex){
		//test passed as the exception was expected
		//do nothing and test will pass.
	}


}
```

### 2. Junit approach

```
//indicate which exception you exception
@Test(expected = ArithmeticException.class)
public void divideThrowsArithmeticExceptionIfDenominatorIsZero() {
		SimpleMath.divide(7, 0);
	}
```

You may still need to user approach 1 if you need to assert on the values of Exception such as message or nested exception

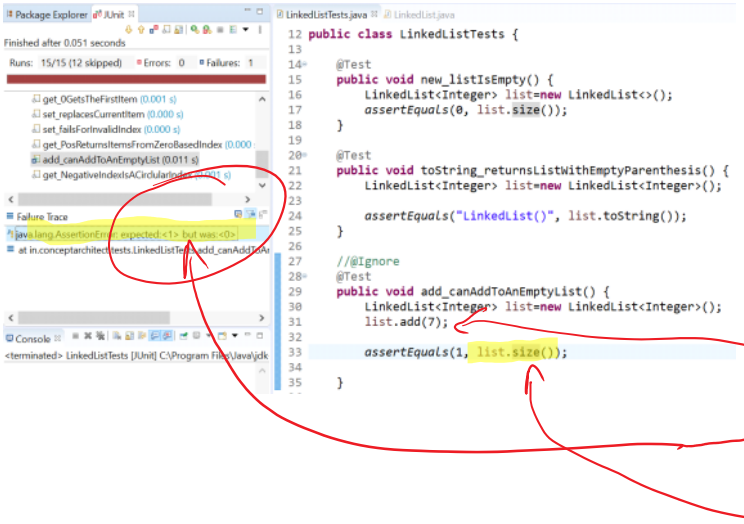
# Assignment

Monday, June 1, 2020 4:35 PM

- Create a jUnit test for LinkedLIst
  - Write test cases for
    - get/set
      - should return value from beginning of list
      - from end of list
      - should throw IndexOutOfBoundException for invalid index
    - add
      - adds to the empty list
      - adds item to the end of non-empty list
    - remove
      - can remove first item
      - can remove last item
      - can remove middle item
    - toString
      - what test can you do with toString?

# Is it a unit test

Tuesday, June 2, 2020 10:31 AM




## Which method are we really testing here?

1. add() or size()?
2. what if there is a logical error in the size()?
   a. Test will still fail.
3. Can a failing test conclusively prove that it's a add() failure and not size() failure?
4. **This is NOT a pure unit test**
   a. **we need to apply two different functions**
5. **It is not always possible to avoid this scenario.**

Why is size 0?

1. add failed to add the item. so size is 0
2. add was successful, but size has a unimplemented logic?

## How do we isolate the problem

1. **(preferred)** the method under test can return a value that can be asserted upon
2. You methods should throw appropriate exceptions on failures and we can assert against failure.
   a. even if function is void and does't throw exception can indicate success
3. make sure we have comrehensive unit test for the helper method ensuring that the other method is working correctly.

# Multiple Asserts

Tuesday, June 2, 2020 10:46 AM

```
@Test
public void add_canAddToAnEmptyList() {
    LinkedList<Integer> list=new LinkedList<Integer>();
    list.add(7);


    assertEquals(1, list.size());
    assertEquals("LinkedList(\t7\t)",list.toString());

}
```

## Q1. How many assert can be present in single test?

**Answer:** There is not limit.

## Q2. How many assert should be present in a single test

**Answer:** There are two school of thoughts

## Answer 2.b Contextual Descision

- Sometimes multiple asserts are mechanism to be double sure of a single fact
  - As it is in the above case
  - We are still doubly verifyng the outcome of adding to an empty list
- This may be a more holestic understanding.
  - logic of size() or toString() may be wrong
  - chances of both being wroing is slim

### Recommendation

- Avoid multiple asserts as much as you.
- They often suggest you don't have a great strategy
- Don't be too strict that you can never have multiple asserts.
- When using multiple asserts, ask yourself if they test one code path only.
- Are they checking the same thing?

## Answer 2.a Strict Rule

- There should be a single assert per test method
- Multiple asserts generally mean you are trying to do test multiple paths in a single test — this vioalates the basic idea of Unit testing
- You should have multiple tests testing all possible outcomes from a single method
  - Example
    - get with valid index
    - get with invalid index
    - get with circular index
- If multiple asserts are allowed test designers may just write one test to test all paths
- When first assertion fails, test fails. It doesn't move forward. So we don't know if others would work or not

```
@Test
public void goodUseCaseOfMultiAssert(){

    list.add(10);          //when I add to an empty list

    assertEquals(1,size());  //list size increased
    assertEquals(10, list.get(0);  //and item becomes the first item

}
```

**This is a good use case, but can we not test thiese two ideas as two separate tests?**

```
@Test
public void badUseCaseOfMultiAssert(){

    list.add(2);
    list.add(9);
    list.add(15);

    assertEquals(2, list.get(0)); //can access 0th item
    assertEquals(15, list.get(2)); //can access last item
    assertEquals(15,list.get(-1)); //circular index is working
    try{
    list.get(100);
    fail("indexoutofbound not thrown");
    }
    catch(IndexOutOfBoundsException ex){

    }
}


}
```

# Test AAA

Tuesday, June 2, 2020 11:04 AM

Every test conceptually follows the idea of AAA

- **Arrange**
  - Prepare for your test
  - Creatign the required objects
  - Add sample data which may be pre-requisite for a test
- **Act**
  - Peform the action which you are about to test
  - Gather the result if required
- **Assert**
  - specify what do you think the ideal response should be

It's a good idea to mark three comments in your test as //Arrange //Act //Assert

## Arrange

- we often need same arrange in multiple tests.
- we can do such initialization at the class level rather than at the method level
- Where should I arrange
  1. In the constructor

### Why I shouldn't arrange in constructor.

- Unit Testing framework follows a life cycle (check **Unit Test Lifecycle page**)

```
@Test
public void add_addedItemsAreShownInToString() {
    //ARRANGE

    //ACT
    list.add(1);
    list.add(2);
    list.add(3);

    //ASSERT
    assertEquals("LinkedList(\t1\t2\t3\t)",
    list.toString());
}
```

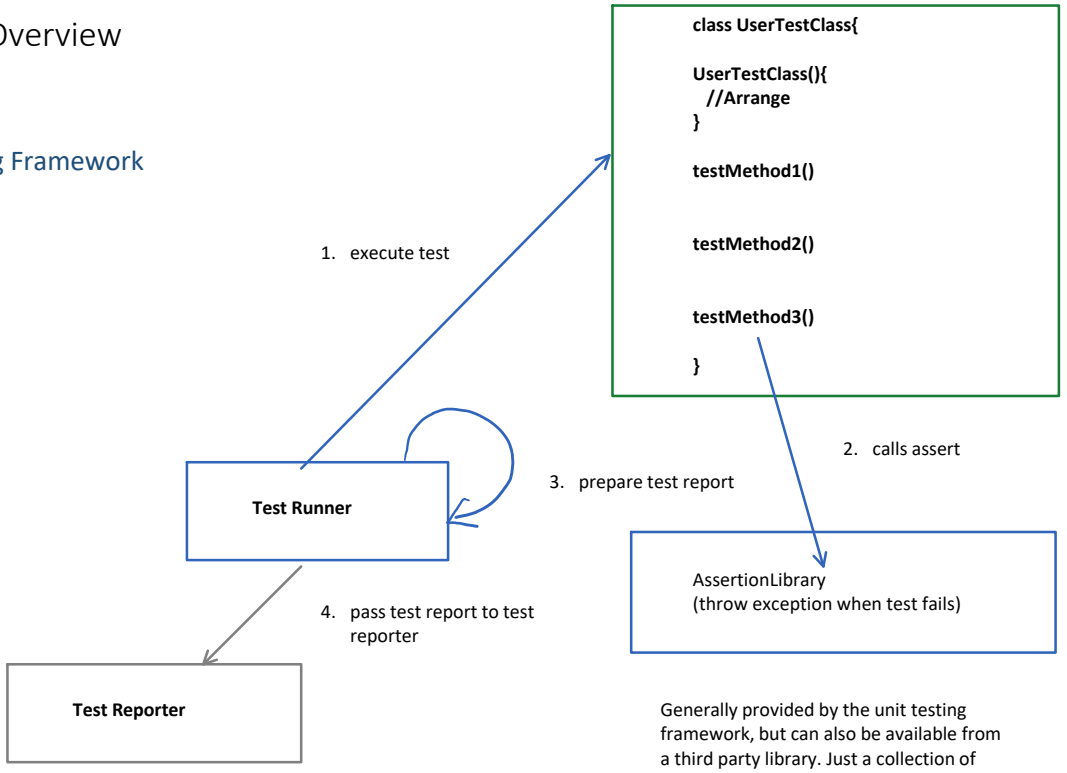
```
@Test
public void get_0GetsTheFirstItem() {
    //ARRANGE
    list.add(10);
    list.add(15);
    list.add(12);

    //ACT

    //ASSERT
}
```

# Unit Test Architectural Overview

Tuesday, June 2, 2020 11:18 AM

## Key Elements in a Unit Testing Framework

1. **Test Runner**
2. User Defined Test Class
3. Assertion Library
4. Test Reporter




Generally provided by the unit testing framework, but can also be available from a third party library. Just a collection of asserts to make your life easy

The Job of test Reporter is to present the test report to user
It can be thirdparty element also. Popular choices include

1. Console. Test framework can put result on console
2. Log file
3. **IDE GUI based test reporter (we use eclipse test reproter)**

## How TestRunner Runs your Unit Test (psudocode to understand the flow)

```
public class TestRunner{

    public List<TestResult>  runTest( ){

        //Step 1. Create object of testClass
        Object testObject= new new LinkedListTest() ;

        ArrayList<TestResult> results=new ArrayList<TestResult>()
        for(TestMethod method :  testObject.getTestMethods())
        {
            //Object testObject=new LinkedListTest(); //Official junit 4 test runner only
            initMethod.execute()
            TestResult result= method.execute();
            results.add(result);
        }
        return result;

    }


}
```

1. *constructor is called only once*
   a. *same list is used by all three methods*
2. each test is called (order not gurateed)

```
class LinkedListTest{
    LinkedList list;
    public void LinkedListTest(){
        list=new LinkedList();
        list.add(1);
        list.add(2);
    }

    @Test
    public void addTest(){   list.add(3); }

    @Test
    public void removeTest( list.remove(0); }

    @Test
    public void sizeTest( list.size(); }
}
```

all tests are using same linkedlist

### what is the size of list in sizeTest()

- Result may vary on order of running
  - result 3  if sequence is
    - addTest-->sizeTest-->removeTest
  - result 2 if sequence is
    - sizeTest-->addTest-->removeTest
    - addTest-->removeTest-->sizeTest
  - result 1 if sequece is
    - removeTest-->sizeTest-->addTest
- Tests are influencing each other and not properly isolated.

## How to Isolate The Test

- To Isolate the test, jUnit provides the concept init method
- **A init method is any method decorated with @Before annotation**

- **Note in the code above the @Before method is called before every running test**
- **Any initialization here ensure that each test gets the same set of data.**

## Constructor Vs Init Method

- A constructor may execute only once before running all test methods
  - Constructor initialization may be shared among different testmethods
  - This may make the design non isolated
- @Before ensures that the method executes before each test
  - It can reset the arrangement
  - One test work doesn't effect others

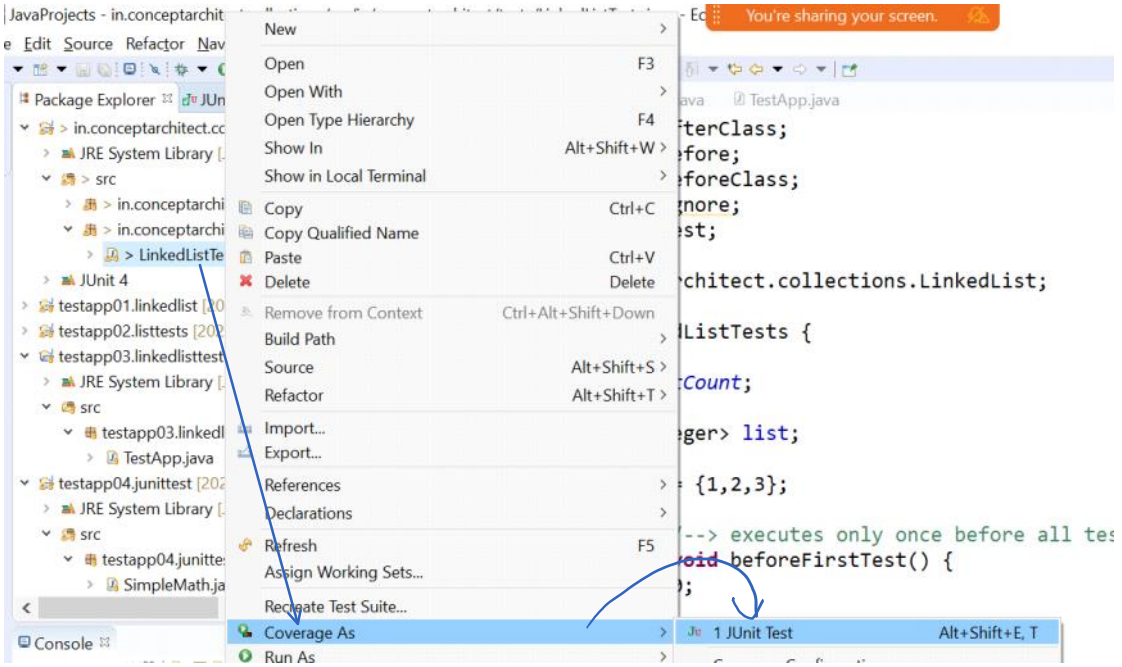
you should avoid constructor initalization and follow @Before initlization

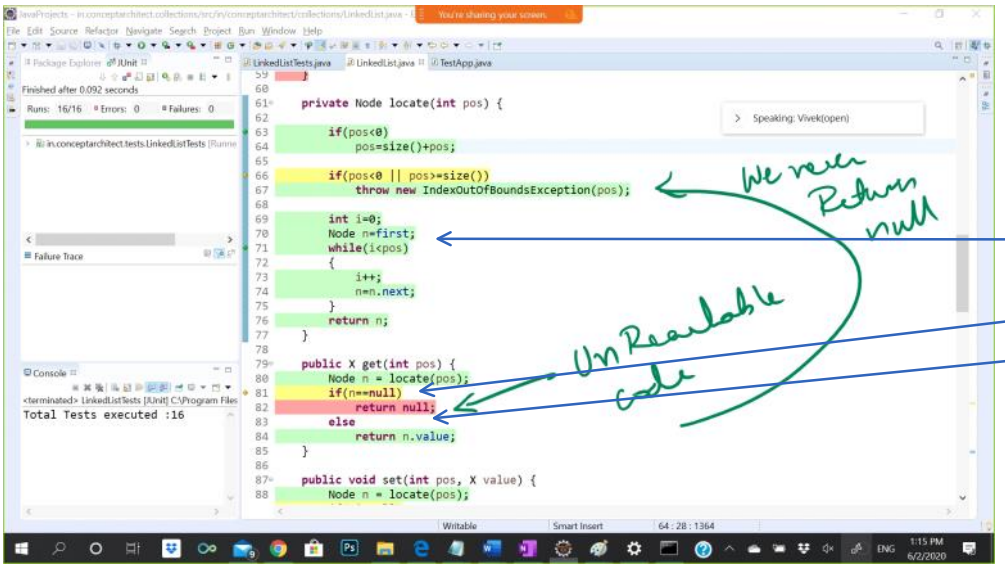
## Junit 4 changes

- Junit 4 onwards, constructor is also called withing the loop
- Now constructor initialzation is also isolated just like @Before
- You can use either of them
- For backward compatiblity and easy readability you should always use @Before
- Many frameworks may replace the default test runner making it possible that constructor approach may fail
-

# code coverage

Tuesday, June 2, 2020 1:16 PM




Color Code

1. Green: This is tested by at least one of the tests
2. Yellow: Partially tested (as never entered the loop)
3. Red. No test reached this spot

## What to do to handle Red code here

- Red code indicate that no test case ever reached here. There are two possible reasons

1. You have not written an important test case to reach there
   a. Think and plan the test to make sure you reach
   b. This should be your first preference
2. That code is actually a unreachable code
   a. Such codes are generally not required
   b. Remove them if you don't need them

## Part Coverage

- Yellow indicate part of the coverage.
- In the given code if is a part coverage
- If has two conditions, probably we never hit one of them
- Solution

```
private Node locate(int pos) {

    if(pos<0)
        pos=size()+pos;

    if(pos<0 || pos>=size())
```

- Yellow indicate part of the coverage.
- In the given code if is a part coverage
- If has two conditions, probably we never hit one of them
- Solution
    1. Write test case to hit the one which is not hit
    2. Delete un-needed condition
- How do

## How do I know which condition is hit?

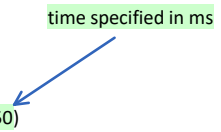
- you may try using two separate if to verify.

```
        if(pos<0)
            pos=size()+pos;

        if(pos<0 || pos>=size())
            throw new IndexOutOfBoundsException(pos);

        int i=0;
        Node n=first;
        while(i<pos)
        {
            i++;
            n=n.next;
        }
        return n;
    }
```

# Timeout Test

Tuesday, June 2, 2020 2:52 PM

- Sometimes we need to make sure that method completes in a given time frame
- If method takes longer than expected, it should be considered a failure




```
@Test(timeout = 50)
public void timeTakenToAccessMaxItems() {
	System.out.println("time taken to access "+max+" items");
	long sum=0;
	for(int i=0;i<list.size();i++) {
		sum+=list.get(i); //it is important we access the item
	}

	System.out.println("Sum of all values is "+sum);

}
```