

static import

Monday, June 1, 2020 5:20 PM

```
package in.conceptarchitect.tests;

//import all static methods from Assert class
//this way all static method of the class 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()
    }
}
```

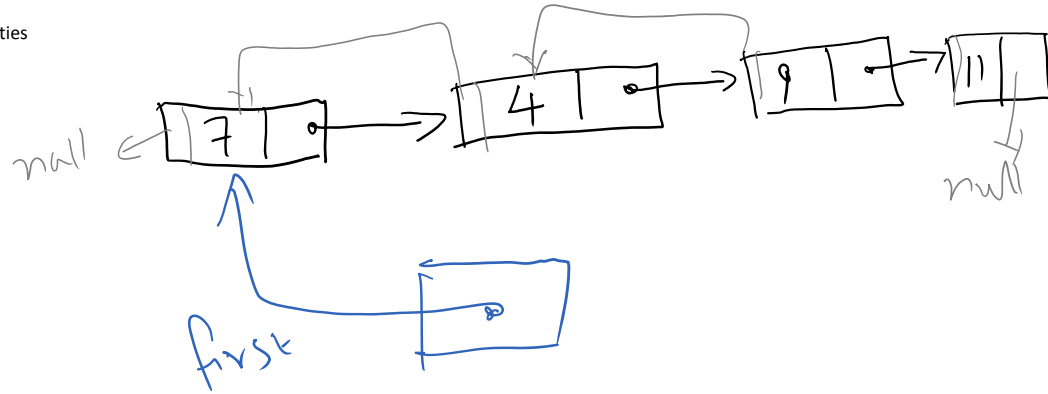
Allows all static method of a class to be imported as a global method

These methods don't require class name to call them

Assignment01 LinkedList

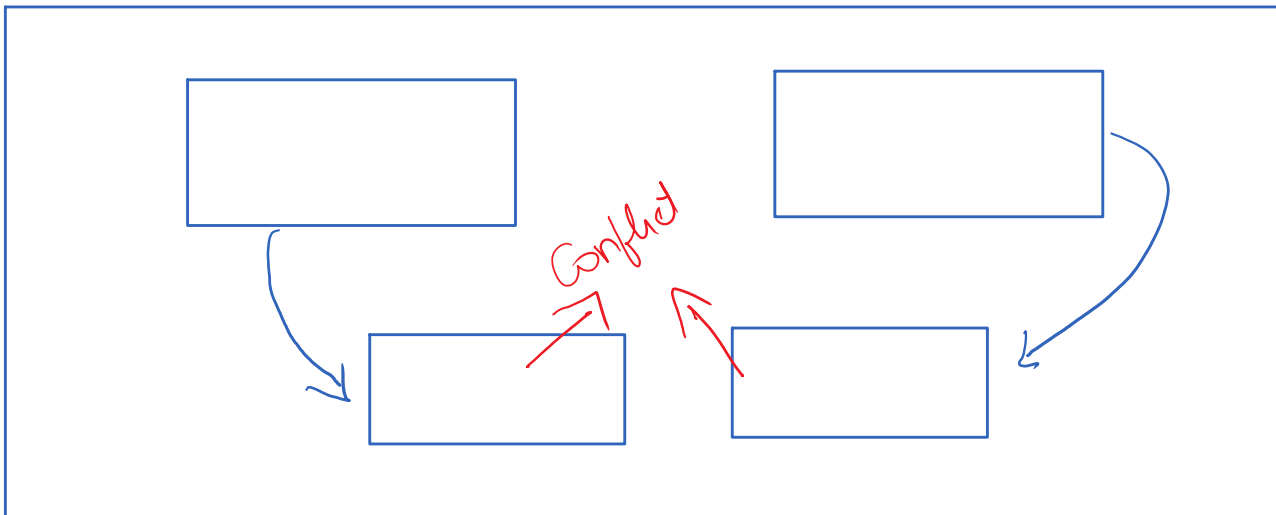
Thursday, May 21, 2020 2:58 PM

- 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 value 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

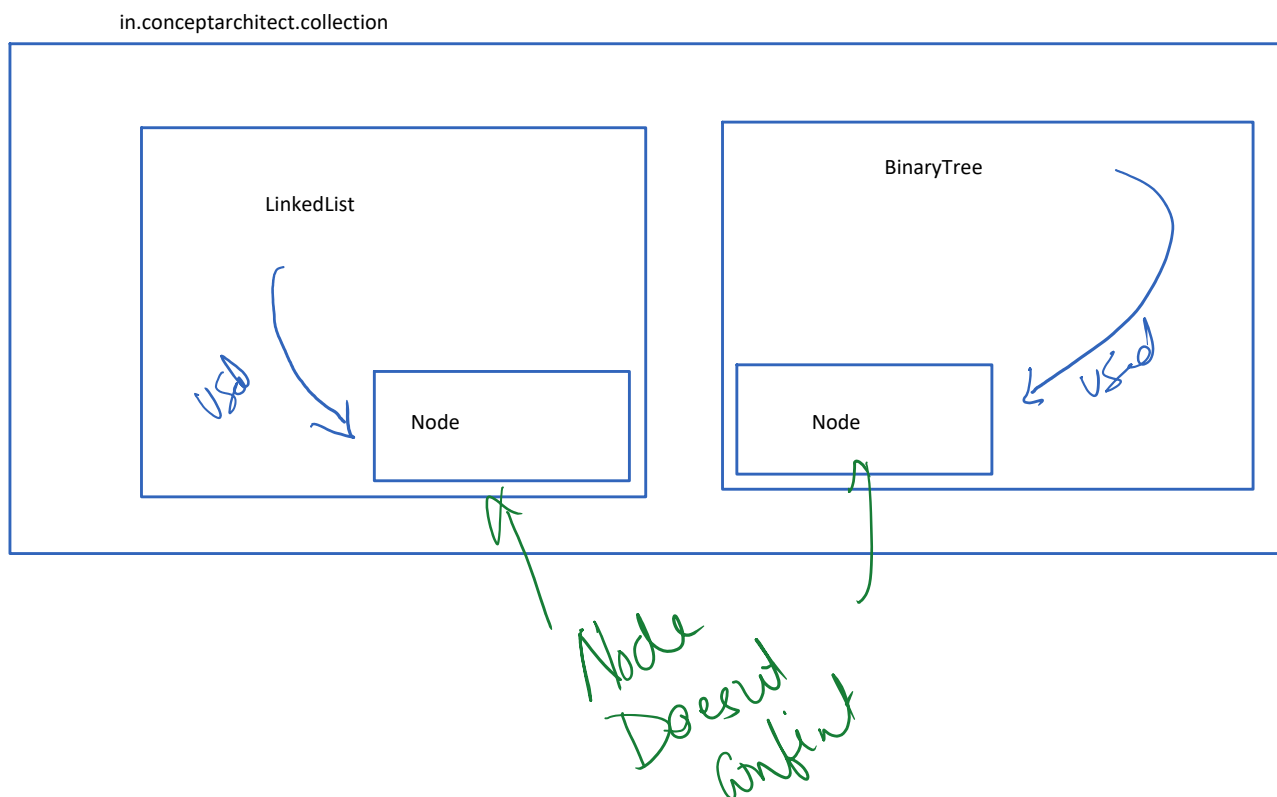


Package vs Class Boundry

Friday, May 22, 2020 3:57 PM



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



When should I use 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

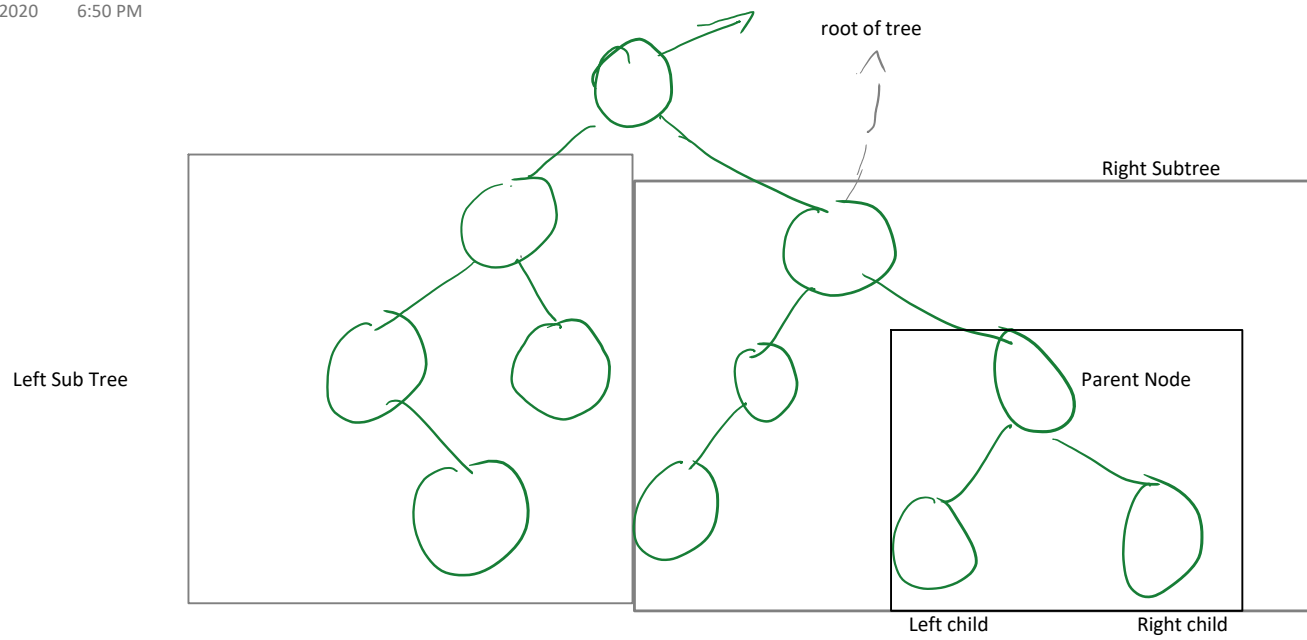
- Make sure, your reusable components that can be productive to more than one applications, should be in its own
 - Package
 - Jar
- A Package is **Not designed to hold a single class**, but it is designed to hold a similar or related set of classes
 - Good Examples
 - collection → to hold collection 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 elementary 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 fictitious 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 throwaway 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

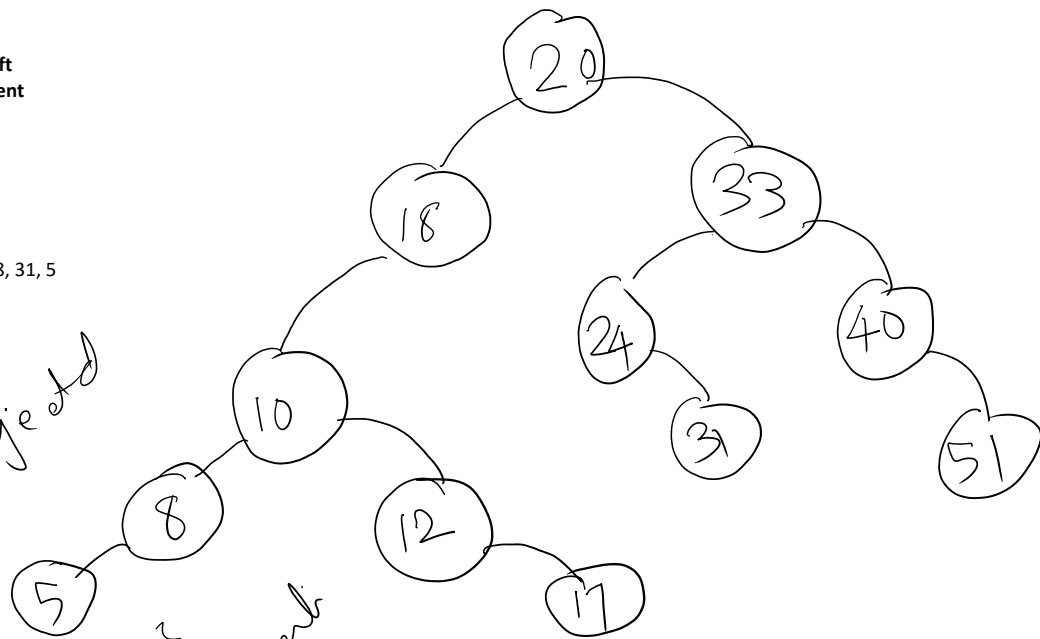
To Add following Numbers

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

↓
Rejected

- If you want to retain duplicate you can change the tree formula to one of the two given below
 - $L \leq P < R$
 - or $L < P \leq R$
- But Not
 - $L \leq P \leq R$

Allow's Duplicate
Neva
Jax



```
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

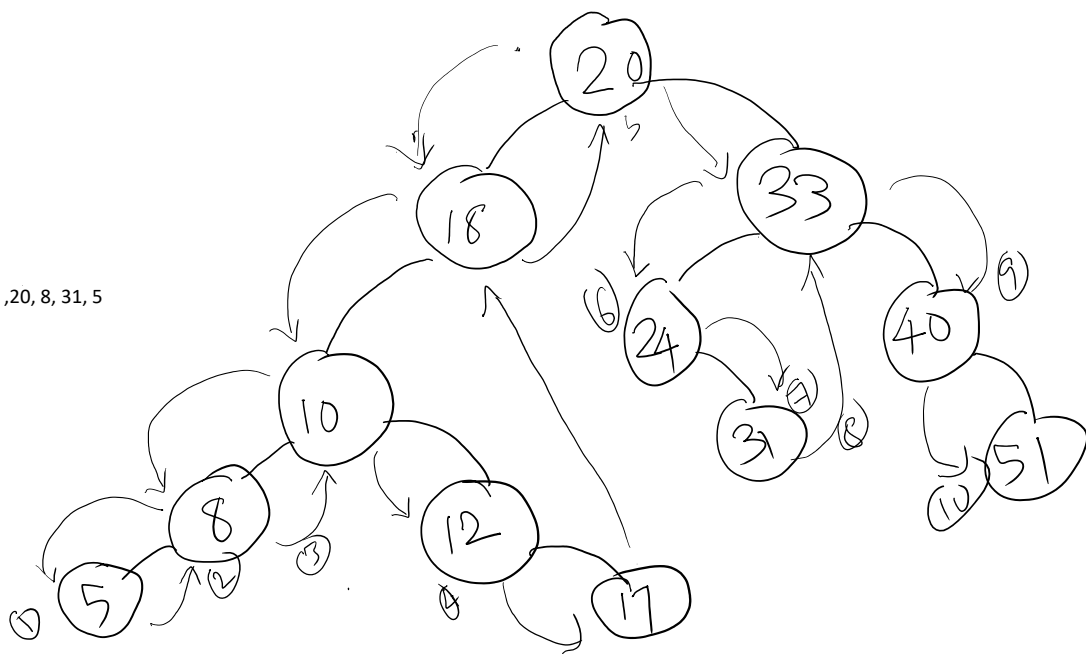
- Visit Left (subtree)
- Visit Parent
- Visit Right(subtree)

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



Preorder

P --> L --> R

Preorder

L --> R --> P

```
Node inorder ( Node root){  
  
    if(root==null){  
        return;  
  
    }  
    inorder(root.left); //L  
    print(root.value); //P  
    inorder(root.right); //R  
  
}
```

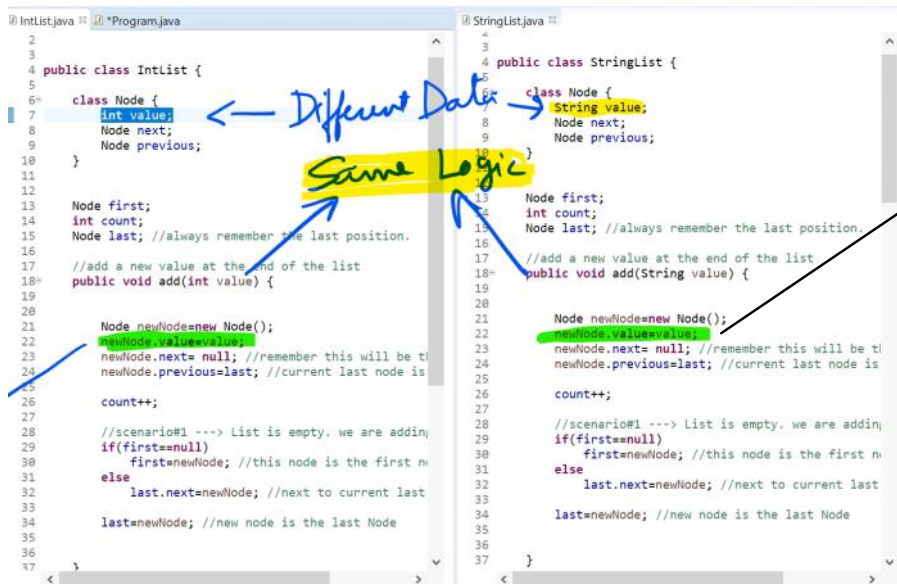

Assignment 02

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



- 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 algorithm easily.

Object List

Monday, June 1, 2020 11:22 AM

```
ObjectList.java
public class ObjectList {
    class Node {
        Object 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(Object value) {
        Node newNode=new Node();
        newNode.value=value;
        newNode.next= null; //remember this will be the last node.
        newNode.previous=last; //current last node is my previous
        count++;
        //scenario1 ----> List is empty. we are adding the first node
        if(first==null)
            first=newNode; //this node is the first node
        else
            last.next=newNode; //next to current last will be newNode
        last=newNode; //new node is the last Node
    }
}

StringList.java
private static void testObjectList() {
    ObjectList newList=new ObjectList();
    newList.add("India");
    newList.add("USA");
    newList.add(21); //PROBLEM #2: This is a list of Object, so any object can be added here
    newList.add("France"); //I can add these values

    for(int i=0;i<newList.size(); i++) {
        //System.out.println(newList.get(i).toUpperCase()); //returned value
        //Problem#1 ----> Manual type casting
        var str= (String) newList.get(i); //PROBLEM#3: Mismatched Error
        System.out.println(str.toUpperCase());
    }

    ObjectList newList=new ObjectList();
    newList.add(20);
    newList.add(30);
    newList.add(15);

    for(int i=0;i<newList.size(); i++) {
        System.out.println(newList.get(i));
    }
}
```

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 Algebraic 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.

```
1 package in.concepts.architect.collections;
2
3
4 //X is some unknown which will be supplied when creating a object
5 public class LinkedList<X> {
6     class Node {
7         X value;
8         Node next;
9         Node previous;
10    }
11    Node first;
12    int count;
13    Node last; //always remember the last position.
14
15    //add a new value at the end of the list
16    public void add(X value) {
17
18        Node newNode = new Node();
19        newNode.value = value;
20        newNode.next = null; //remember this will be the last node.
21        newNode.previous = last; //current last node is my previous
22
23        count++;
24
25        //scenario1 ---> List is empty. we are adding the first node
26        if(first == null) {
27            first = newNode; //this node is the first node
28        }
29        else {
30            last.next = newNode; //next to current last will be newNode
31            last = newNode; //new node is the last Node
32        }
33    }
34 }
35
36
```

```
16 //TestOtringList()
17
18 //Using ObjectList to strings
19 //testObjectList()
20
21 LinkedList<String> names = new LinkedList<String>();
22 names.add("India");
23 names.add("USA");
24 //names.add(23); //PROBLEM SOLVED #2: Invalid Data is rejected
25 names.add("France"); //I can add these values
26
27
28 for(int i=0; i<names.size(); i++) {
29     //System.out.println(names.get(i).toUpperCase()); //returns v
30
31     //Problem#1 Solved --> No Typecasting need
32     String str = names.get(i).toUpperCase();
33     //PROBLEM#3 SOLVED No Typecasting no question of error Misplace
34
35     System.out.println(str);
36 }
37
38 //If we don't specify the generic type
39 //It is assumed to be an Object
40 //But it is not a recommended practice
41 //It was don't for backward compatibility
42
43 LinkedList numbers = new LinkedList();
44 numbers.add(20);
45 numbers.add(30);
46 numbers.add(15);
47
48 for(int i=0; i<numbers.size(); i++) {
49     System.out.println(numbers.get(i));
50 }
51
```

The benefits

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

Java Implementation

- Java allows to create object without specifying 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 feature 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
 - Intellisense 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= 1; //—> 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

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() output you must look and verify if the result is expected or not
 - system can't decide for you
 - 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

```
5 public class TestApp {
6
7-   public static void main(String[] args) {
8       // TODO Auto-generated method stub
9
10      //LinkedList<int> numbers=new LinkedList<>(); /
11      LinkedList<Integer> numbers=new LinkedList<>();
12      System.out.println(numbers);
13
14      //Test the linkedlist Add
15      testAdd(numbers);
16
17      testGet(numbers);
18
19      testSet(numbers);
20
21      testDelete(numbers);
22  }
23
24  private static void testDelete(LinkedList<Integer>
25      // TODO Auto-generated method stub
26      int [] positions= {8, 6, 2, 0};
27      for(int position : positions) {
28          System.out.println("trying to delete at pos
29          numbers.remove(position);
30      }
31  }
```

Console Output:

```
<terminated> TestApp [Java Application] C:\Program Files\Java\jdk-10.0.2\bin\java.exe (Jun 1, 2020, 12:47:30 PM)
LinkedList()
numbers.size()=10
LinkedList( 0 1 2 3 4 5)
numbers.get(6) is 6
numbers.get(4) is 4
numbers.get(9) is 9
numbers.get(0) is 0
setting numbers.set(6,60) ...
setting numbers.set(4,40) ...
setting numbers.set(9,90) ...
setting numbers.set(0,0) ...
LinkedList( 0 1 2 3 40 5)
trying to delete at position 8
trying to delete at position 6
trying to delete at position 2
trying to delete at position 0
LinkedList( 1 3 40 5 7 90)
```

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

```
11 LinkedList<Integer> numbers=new LinkedList<>();
12 System.out.println(numbers);
13
14 //Test the linkedlist Add
15 testAdd(numbers);
16
17 testGet(numbers);
18
19 testSet(numbers);
20
21 testDelete(numbers);
22
23
24 private static void testGet(LinkedList<Integer> num
25 // TODO Auto-generated method stub
26 int [] positions= {6, 4, 9, 0};
27 for(int position : positions) {
28     int value=numbers.get(position);
29     System.out.print("numbers.get("+position+")
30     if(value==position)
31         System.out.println("t passed");
32     else
33         System.out.println("t failed");
34 }
35 }
```

Console Output:

```
<terminated> TestApp [Java Application] C:\Program Files\Java\jdk-10.0.2\bin\java.exe (Jun 1, 2020, 1:08:00 PM)
LinkedList()
numbers.size()=10
LinkedList( 0 1 2 3 4 5)
numbers.get(6) is 6 passed
numbers.get(4) is 4 passed
numbers.get(9) is 9 passed
numbers.get(0) is 0 passed
setting numbers.set(6,60) ...
setting numbers.set(4,40) ...
setting numbers.set(9,90) ...
setting numbers.set(0,0) ...
LinkedList( 0 1 2 3 40 5)
trying to delete at position 8
trying to delete at position 6
trying to delete at position 2
trying to delete at position 0
LinkedList( 1 3 40 5 7 90)
```

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 correctly?
 - 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 before calling set?
 - e. Result is changing based on call sequence

```
1 package testapp03.linkedlisttests;
2
3 import in.conceptarchitect.collections.LinkedList;
4
5 public class TestApp {
6
7-   public static void main(String[] args) {
8       // TODO Auto-generated method stub
9
10      //LinkedList<int> numbers=new LinkedList<>(); /
11      LinkedList<Integer> numbers=new LinkedList<>();
12      System.out.println(numbers);
13
14      //Test the linkedlist Add
15      testAdd(numbers);
16
17      testSet(numbers);
18
19      testGet(numbers);
20
21      testDelete(numbers);
22  }
23
24  private static void testGet(LinkedList<Integer> num
25      // TODO Auto-generated method stub
26      int [] positions= {6, 4, 9, 0};
27      for(int position : positions) {
28          int value=numbers.get(position);
29          System.out.print("numbers.get("+position+")
30          if(value==position)
31              System.out.println("t passed");
32          else
33              System.out.println("t failed");
34      }
35  }
```

Console Output:

```
<terminated> TestApp [Java Application] C:\Program Files\Java\jdk-10.0.2\bin\java.exe (Jun 1, 2020, 1:00:05 PM)
LinkedList()
numbers.size()=10
LinkedList( 0 1 2 3 4 5)
setting numbers.set(6,60) ...
setting numbers.set(4,40) ...
setting numbers.set(9,90) ...
setting numbers.set(0,0) ...
LinkedList( 0 1 2 3 40 5)
numbers.get(6) is 60 failed
numbers.get(4) is 40 failed
numbers.get(9) is 90 failed
numbers.get(0) is 0 passed
trying to delete at position 8
trying to delete at position 6
trying to delete at position 2
trying to delete at position 0
LinkedList( 1 3 40 5 7 90)
```

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:

```

TestApp.java  LinkedList.java
14 //Test the linkedlist Add
15 testAdd(numbers);
16 testSet(numbers);
17 testGetWithInvalidIndex(numbers);
18 testGet(numbers);
19 testDelete(numbers);
20 }
21
22 private static void testGetWithInvalidIndex(LinkedList<Integer> numbers) {
23 // TODO Auto-generated method stub
24 System.out.println("numbers.get(100) is "+numbers.get(100));
25 }
26
27 private static void testGet(LinkedList<Integer> numbers) {
28
29 }
30 }

```

```

Console
<terminated> TestApp [Java Application] C:\Program Files\Java\jdk-10.0.2\bin\javaw.exe (Jun 1, 2020, 1:19:51 PM - 1:19:52 PM)
LinkedList()
numbers.size()=10
LinkedList( 0 1 2 3 4 5 6 7 8 9 )
setting numbers.set(6,60) ...
setting numbers.set(4,40) ...
setting numbers.set(9,90) ...
setting numbers.set(0,0) ...
LinkedList( 0 1 2 3 40 5 60 7 8 90 )
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index out of range: 100
at in.conceptarchitect.collections.LinkedList.locate(LinkedList.java:62)
at in.conceptarchitect.collections.LinkedList.get(LinkedList.java:75)
at testapp03.linkedlisttests.TestApp.testGetWithInvalidIndex(TestApp.java:29)
at testapp03.linkedlisttests.TestApp.main(TestApp.java:19)

```

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

```

TestApp.java  LinkedList.java
14 //Test the linkedlist Add
15 testAdd(numbers);
16
17 testDelete(numbers);
18
19 testGet(numbers);
20
21
22 testSet(numbers);
23 testGetWithInvalidIndex(numbers);
24 }
25
26 private static void testGetWithInvalidIndex(LinkedList<Integer> numbers) {
27 // TODO Auto-generated method stub
28 System.out.println("numbers.get(100) is "+numbers.get(100));
29 }
30 }

```

```

Console
<terminated> TestApp [Java Application] C:\Program Files\Java\jdk-10.0.2\bin\javaw.exe (Jun 1, 2020, 1:30:55 PM - 1:30:56 PM)
LinkedList()
numbers.size()=10
LinkedList( 0 1 2 3 4 5 6 7 8 9 )
trying to delete at position 8
trying to delete at position 6
trying to delete at position 2
trying to delete at position 0
LinkedList( 1 3 4 5 7 9 )
Exception in thread "main" java.lang.NullPointerException
at testapp03.linkedlisttests.TestApp.testGet(TestApp.java:34)
at testapp03.linkedlisttests.TestApp.main(TestApp.java:19)

```

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 boundaries 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

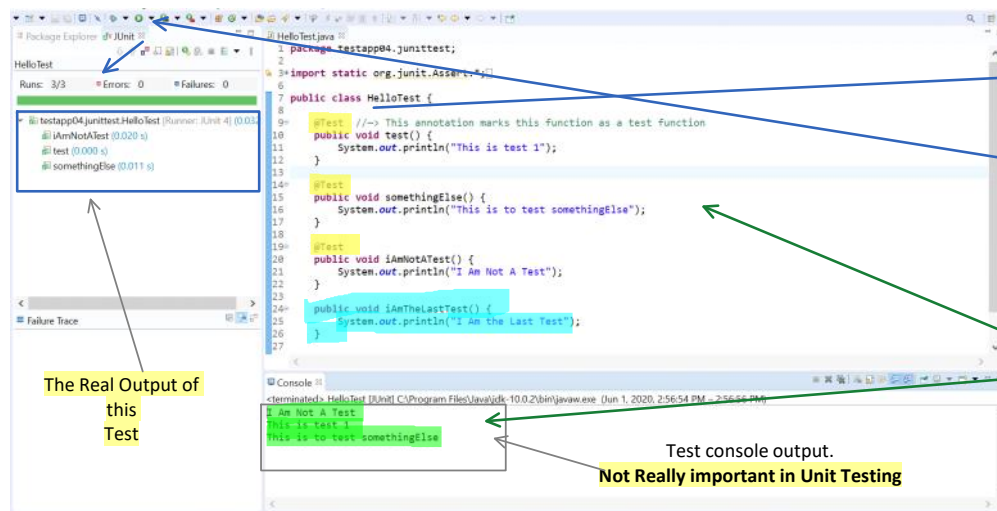
1. Automatic
 - a. Can detect if the test is giving correct result or not
 - i. Not based on `main()` and `print()`
2. Atomic
 - a. Each test is expected to test a very small atomic unit of the code and ensuring this piece works
3. Isolated
 - a. Tests should not influence each other. They all should 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 influenced the design of testing frameworks across all programming languages.

JUnit Test Design

Monday, June 1, 2020 2:57 PM



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

The Real Output of this Test

Test console output.
Not Really important in Unit Testing

Test Explorer

Monday, June 1, 2020 3:09 PM

Test Explorer

Package Explorer JUnit

HelloTest

Runs: 3/3 Errors: 0 Failures: 0

testapp04.junittest.HelloTest [Runner: JUnit 4] (0.032 s)

- AmNotATest (0.020 s)
- test (0.000 s)
- somethingElse (0.011 s)

Failure Trace

Test Summary

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

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)

Tick against individual tests

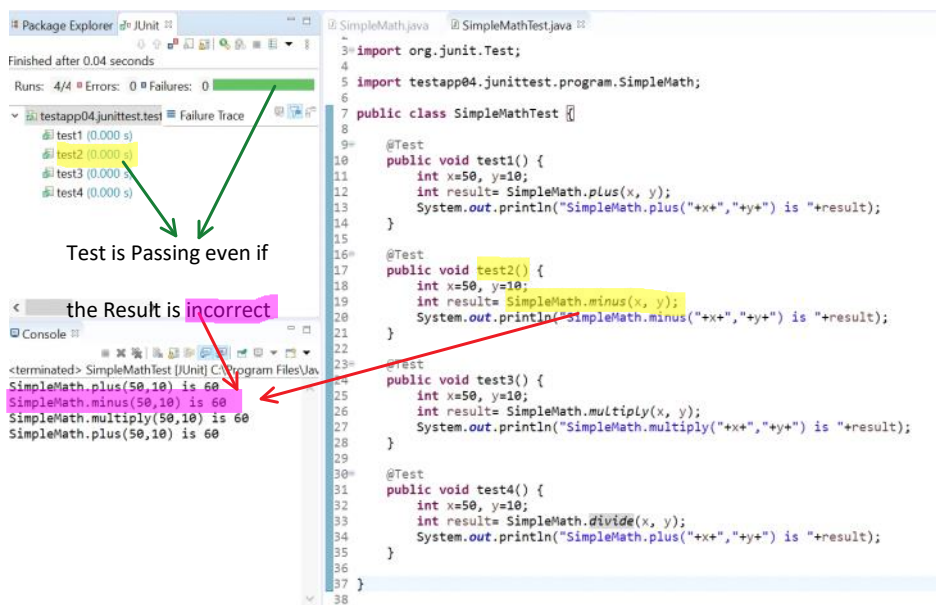
- green indicates success

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

Overall Result

A passing test.

2 tests with errors

1. Failed Test

```
SimpleMathTest.java
12 int x=50, y=10;
13 int result= SimpleMath.plus(x, y);
14 System.out.println("SimpleMath.plus("+x+", "+y+") is "+result);
15 if(result!=x+y)
16     throw new RuntimeException("Plus Operation Failed");
17 }
18
19 @Test
20 public void test2() {
21     int x=50, y=10;
22     int result= SimpleMath.minus(x, y);
23     System.out.println("SimpleMath.minus("+x+", "+y+") is "+result);
24     if(result!=x-y)
25         throw new RuntimeException("Minus Operation Failed");
26 }
27
28 @Test
29 public void test3() {
30     int x=50, y=10;
31     int result= SimpleMath.multiply(x, y);
32     System.out.println("SimpleMath.multiply("+x+", "+y+") is "+result);
33     if(result!=x*y)
34         throw new AssertionError("Multiply Operation Failed");
35 }
36
37 @Test
38 public void test4() {
39     int x=50, y=10;
40     int result= SimpleMath.divide(x, y);
41     System.out.println("SimpleMath.divide("+x+", "+y+") is "+result);
42     if(result!=x/y)
43         throw new RuntimeException("Divide Operation Failed");
44 }
```

What is the difference between an Error and A failure

- The purpose of a unit test is to identify 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

Important

AssertionFailedError: Failed -- Expected 500 actual 60
nitest.tests.SimpleMathTest.isEqual(SimpleMathTest.java:41)
nitest.tests.SimpleMathTest.test3(SimpleMathTest.java:36)

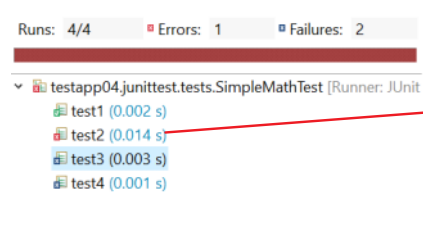
Not Important

```
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000
```

We can have Test Helper
that throws
AssertionFailedError
in case the expected
condition is not me

Test Design Practices

Monday, June 1, 2020 4:01 PM



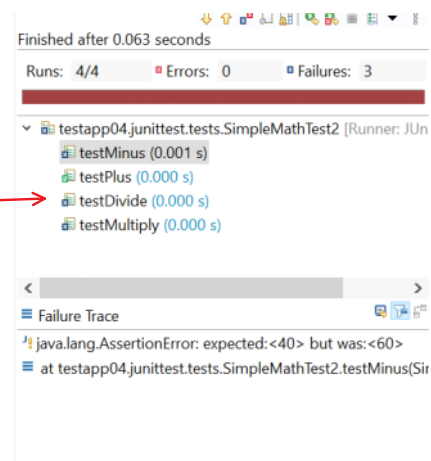
What does test2 do?

- 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

```
@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 AssertionError {
    if(actual!=expected)
        throw new AssertionError("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. `isEqual` 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()

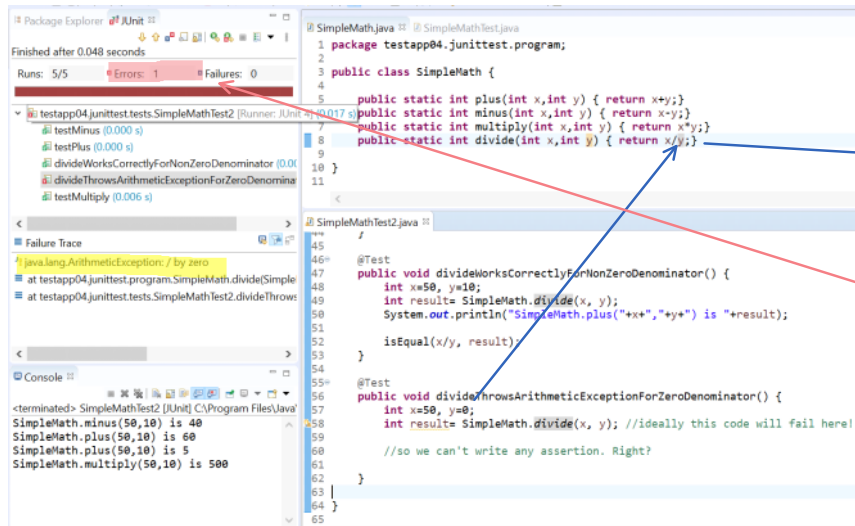
A test name should follow **DAMP** principle

DAMP --> Descriptive and meaningful phrases

- Normally you method names should be meaningful words.
- Your test methods should be longer and descriptive phrases or sentences not just words

Asserting For Exception

Monday, June 1, 2020 4:22 PM



If $y \neq 0$ 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 ArithmeticException

        //If I reach here. It means exception is not throw and
        //the test has failed
        fail("expected 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

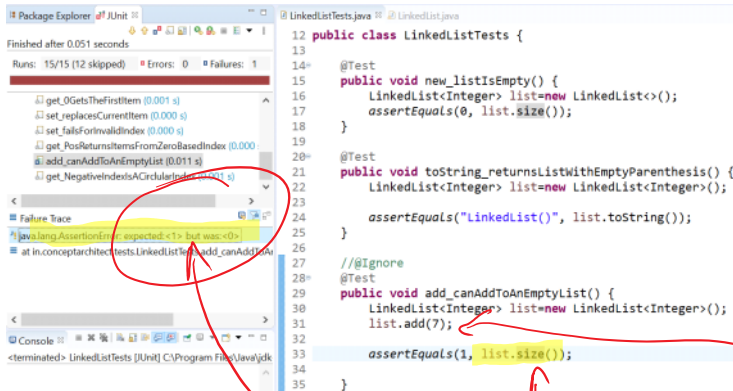
Monday, June 1, 2020

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 IndexOutOfBoundsException 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. make sure we have comprehensive 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());  
}
```

Answer 2.b Contextual Decision

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

@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 these two ideas as two separate tests?

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.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 violates 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 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
 - Create the required objects
 - Add sample data which may be pre-requisite for a test
- **Act**
 - Perform 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

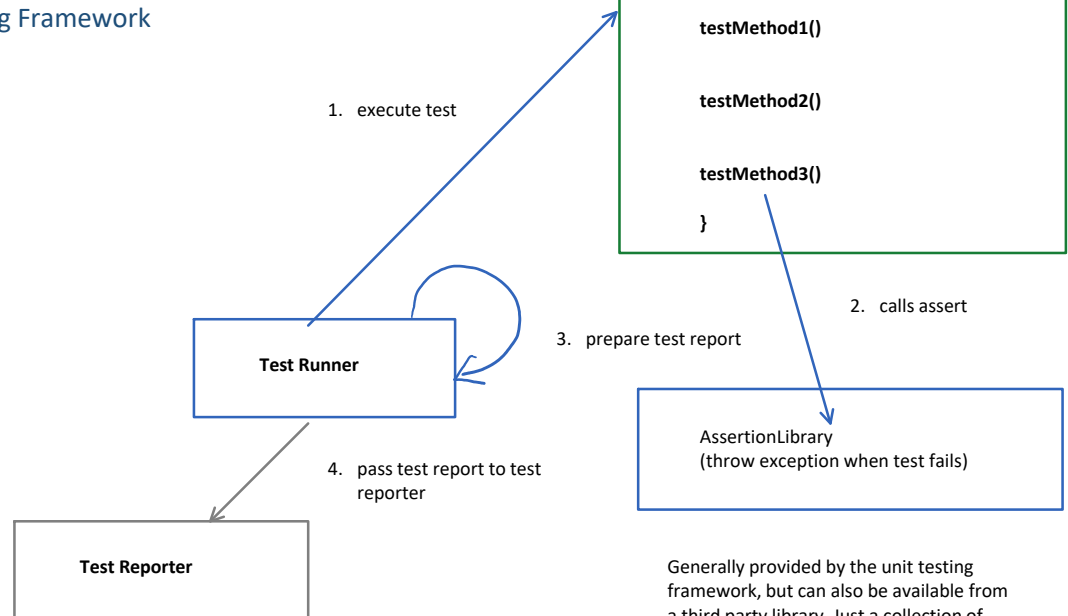
    //ASSERT
}
```

Unit Test Architectural Overview

Tuesday, June 2, 2020 11:18 AM

Key Elements in a Unit Testing Framework

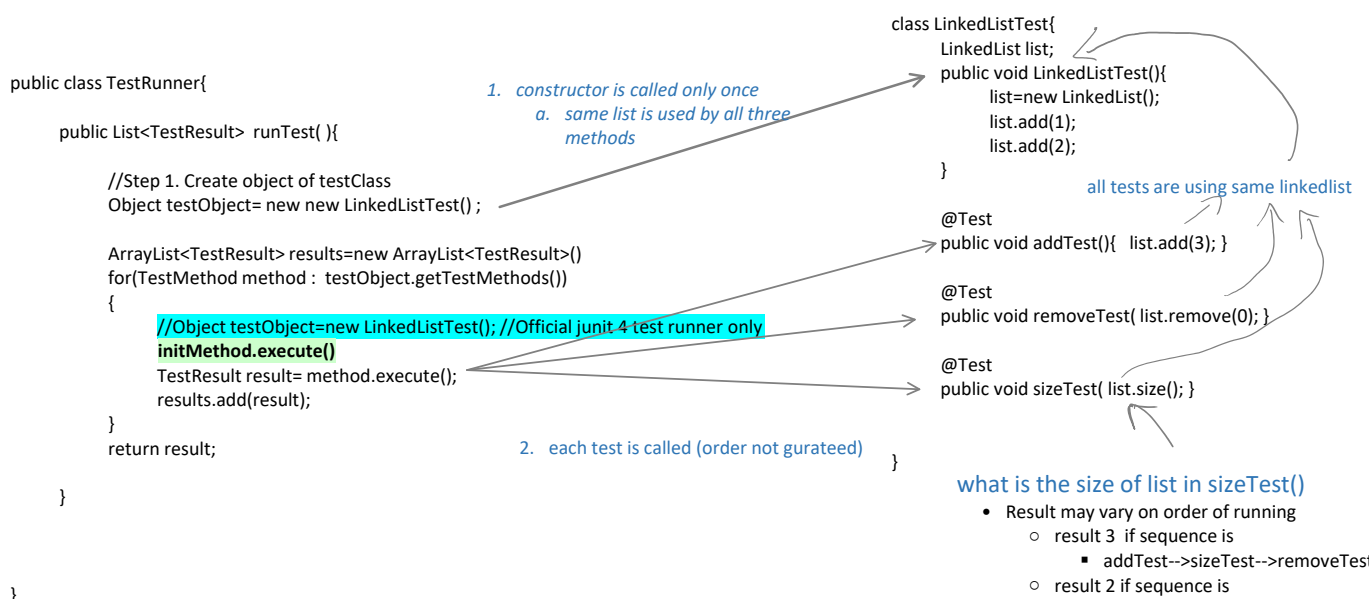
1. Test Runner
2. User Defined Test Class
3. Assertion Library
4. Test Reporter



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 reporter)

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



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 initialization and follow @Before initialization

JUnit 4 changes

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