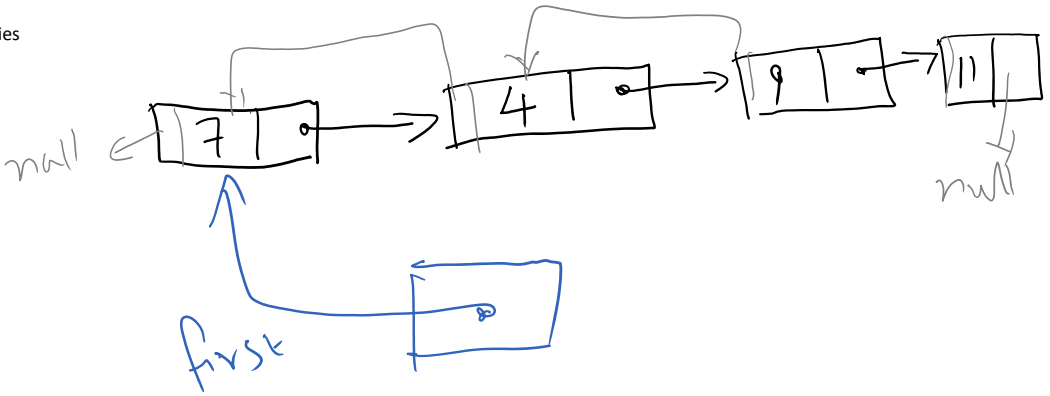
Assignment01 LinkedList

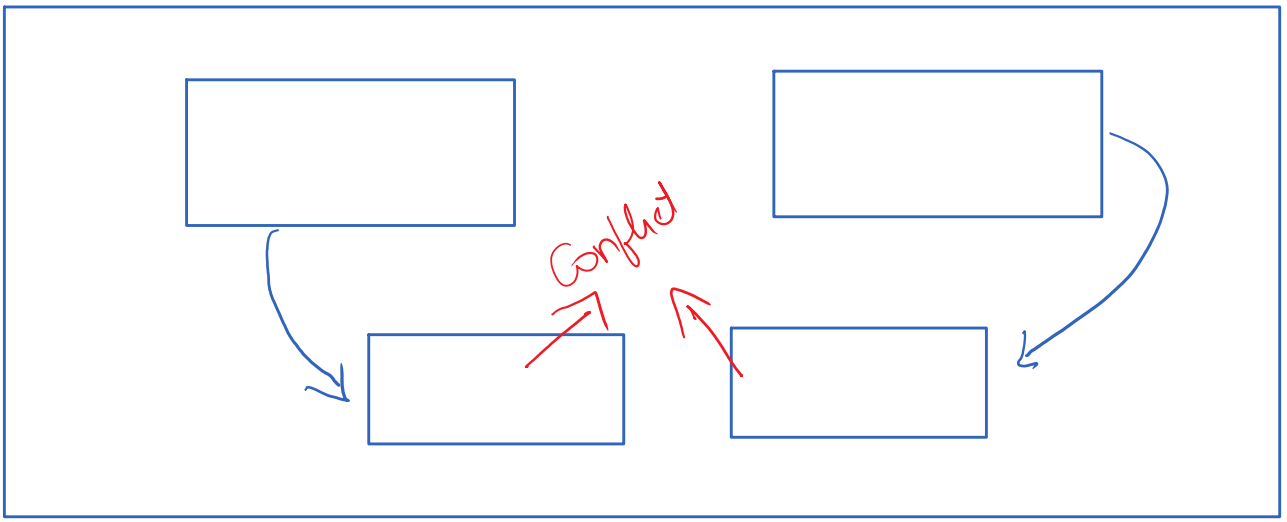
Thursday, May 21, 2020 2:58 PM

- 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

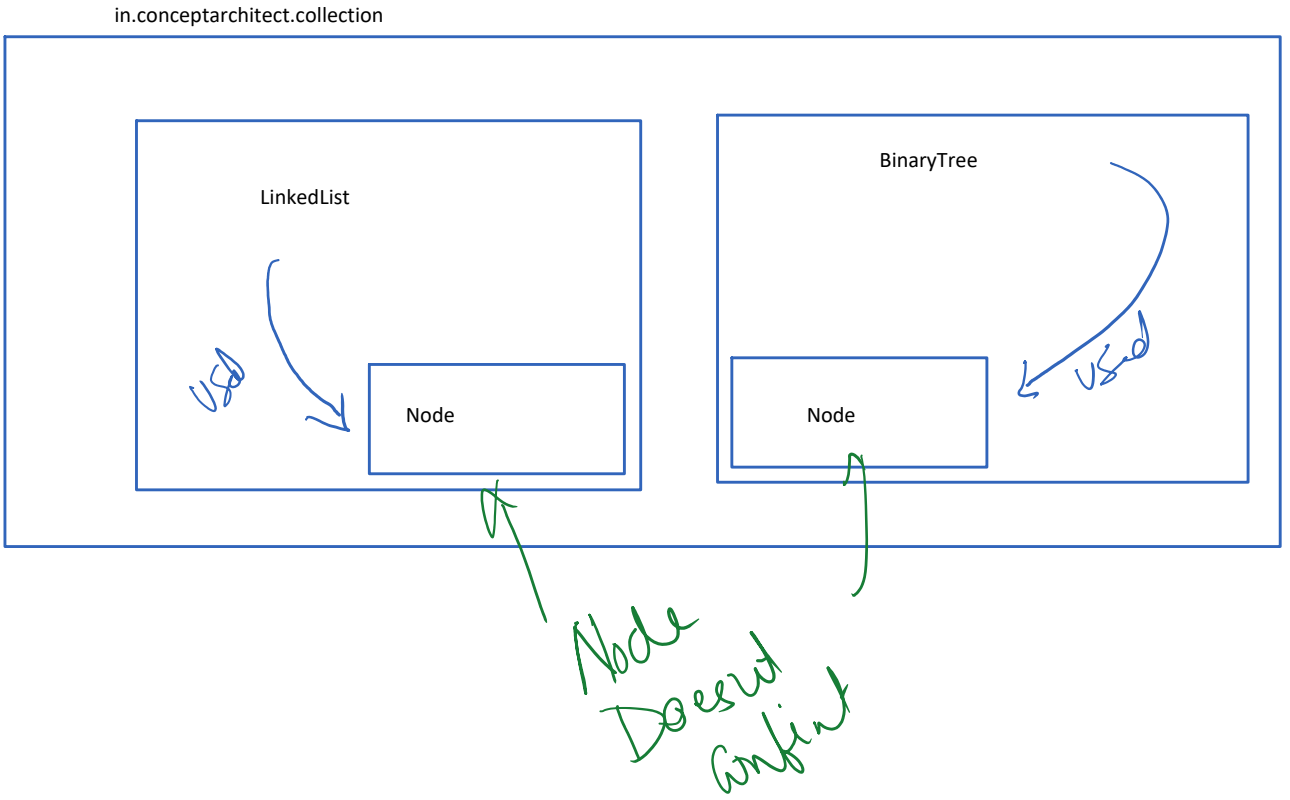


Package vs Class Boundry

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

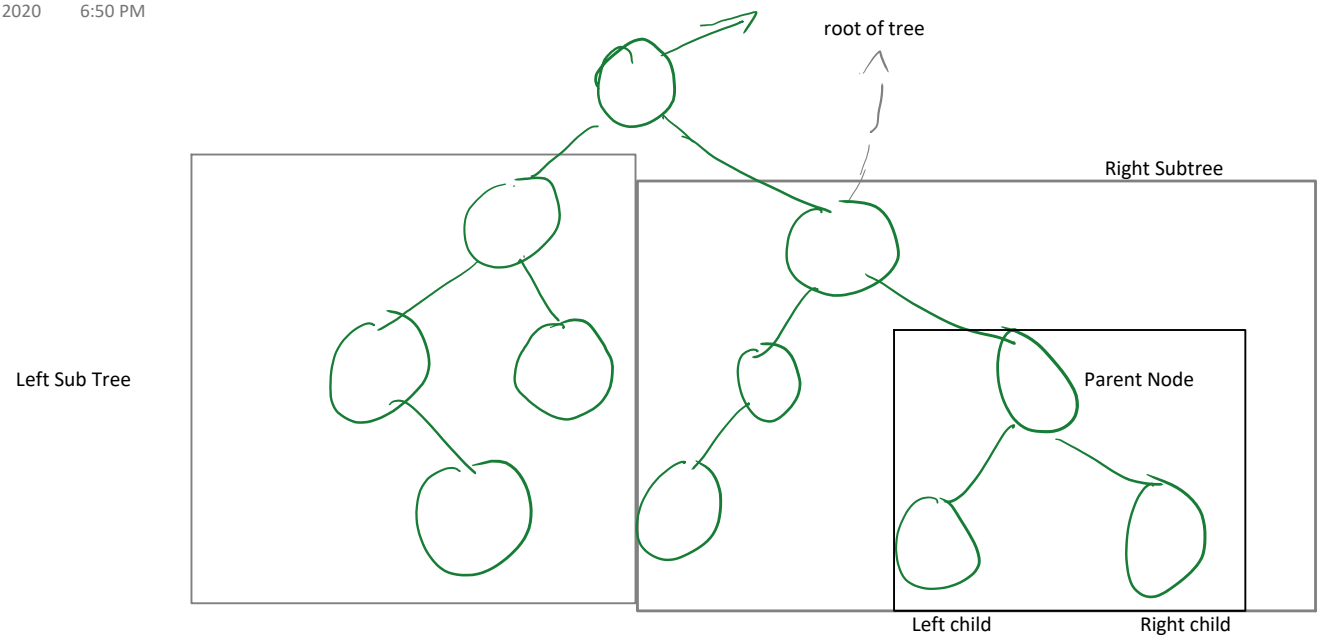
Don'ts

- 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

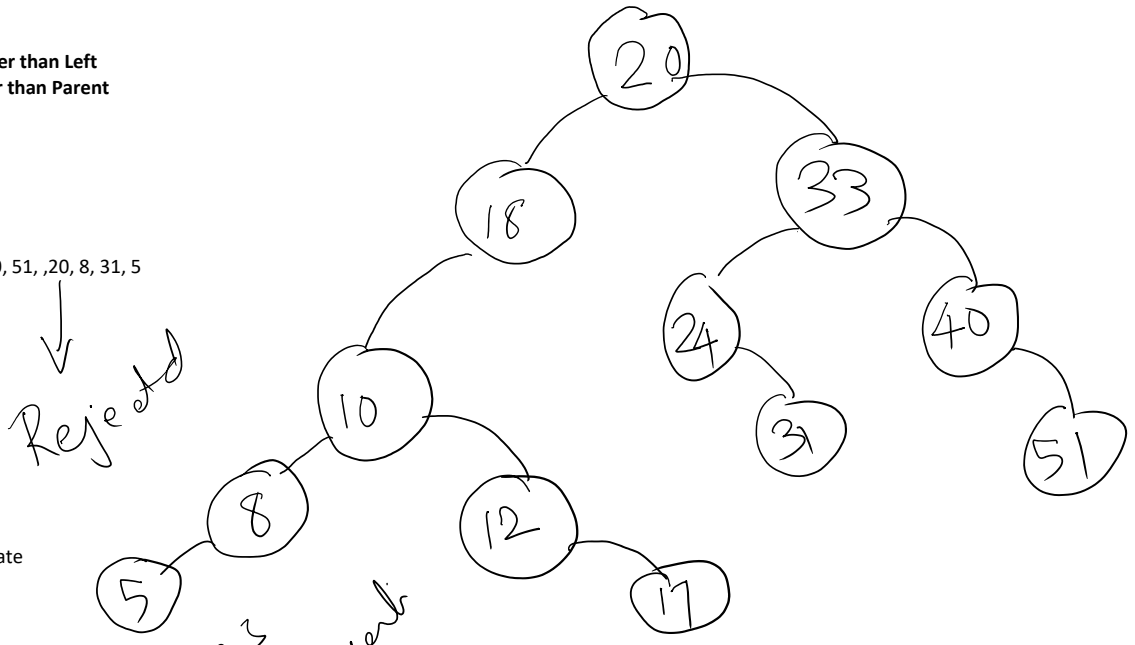
Friday, May 22, 2020

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

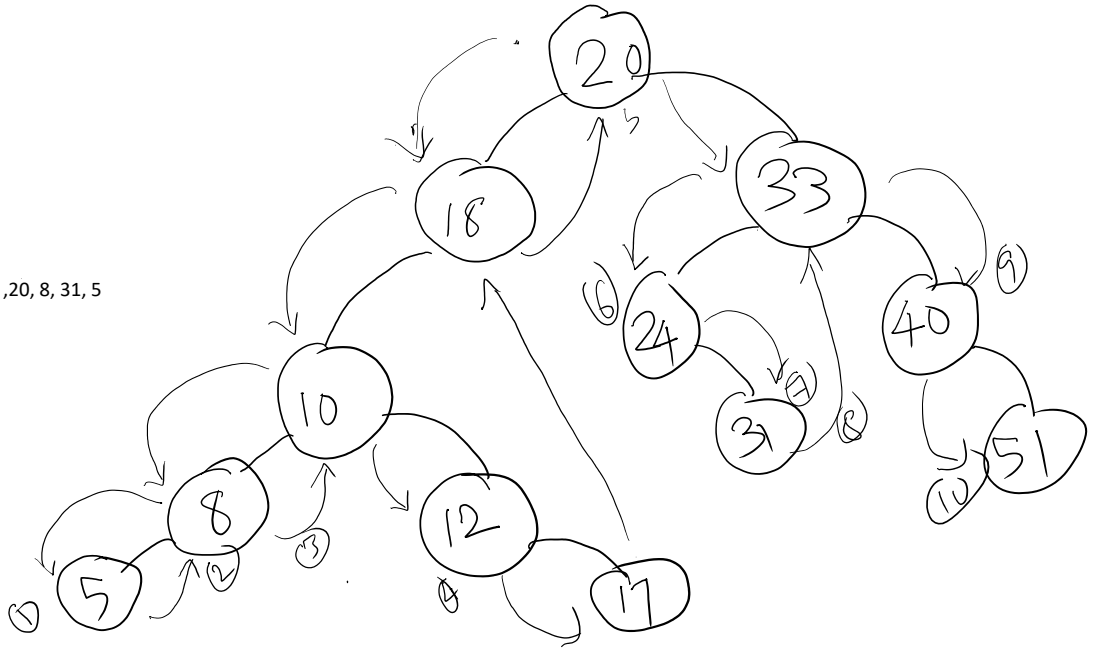

```
Node  insert ( Node root, int value){

        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

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

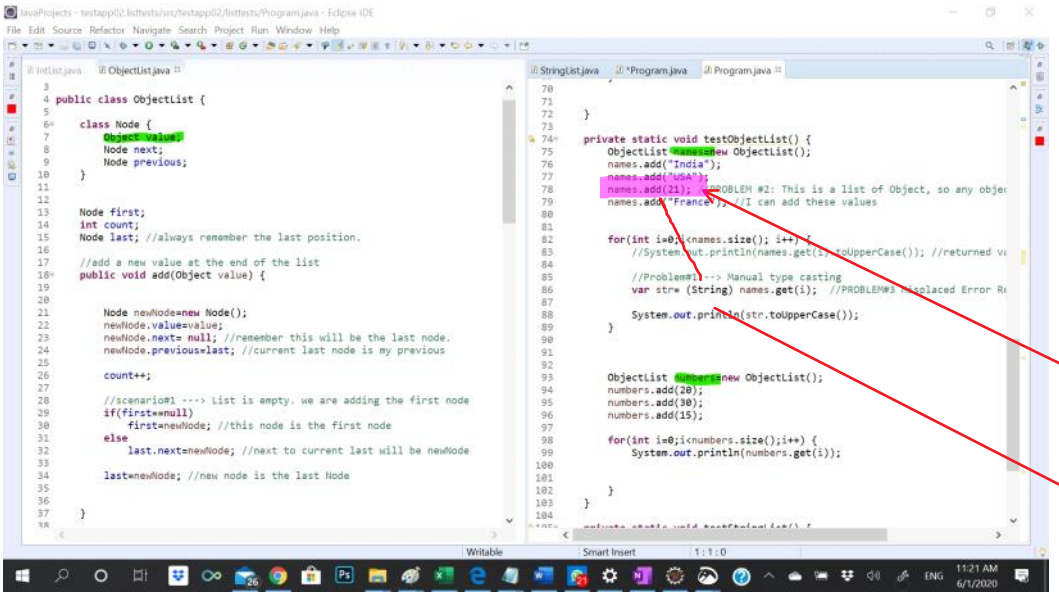

```
public class IntList {

    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
-

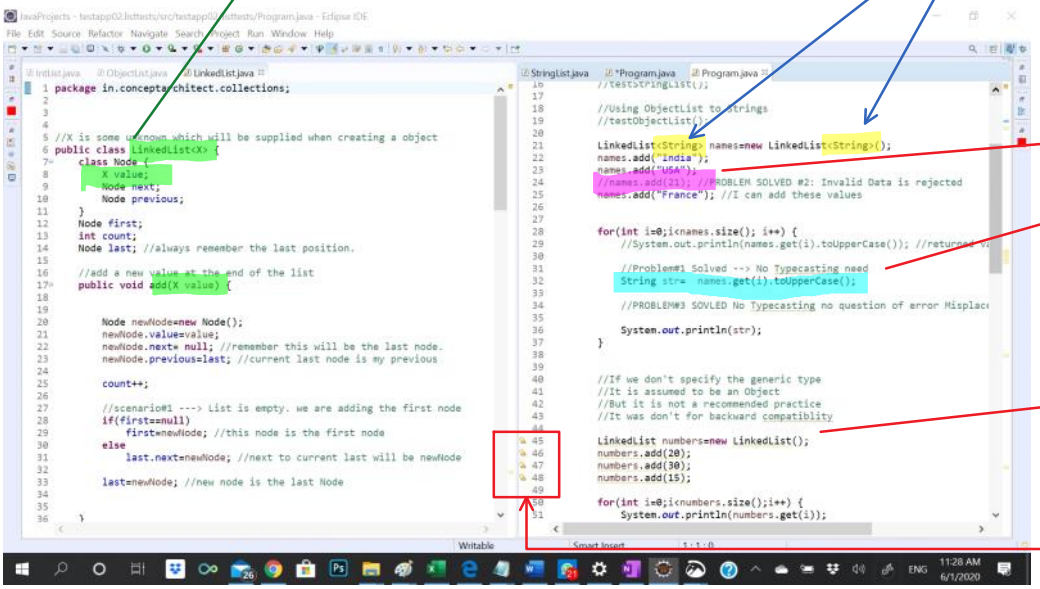
Generics

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

Print and Main Based Test

Monday, June 1, 2020 12:56 PM

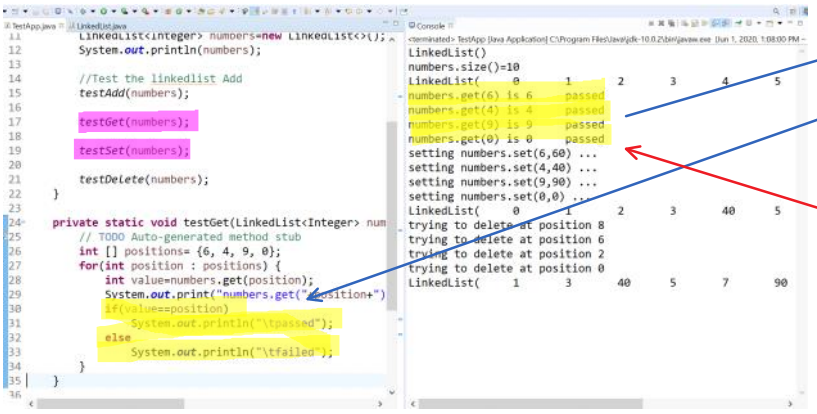
main() function wasn't designed to test your code. It was to run a tested code

- 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

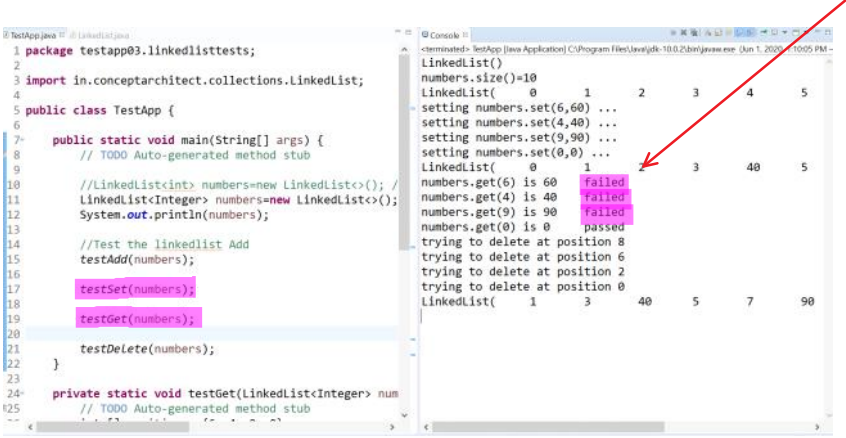


Test Results can be interpreted better

if there is a good logic written

Problem

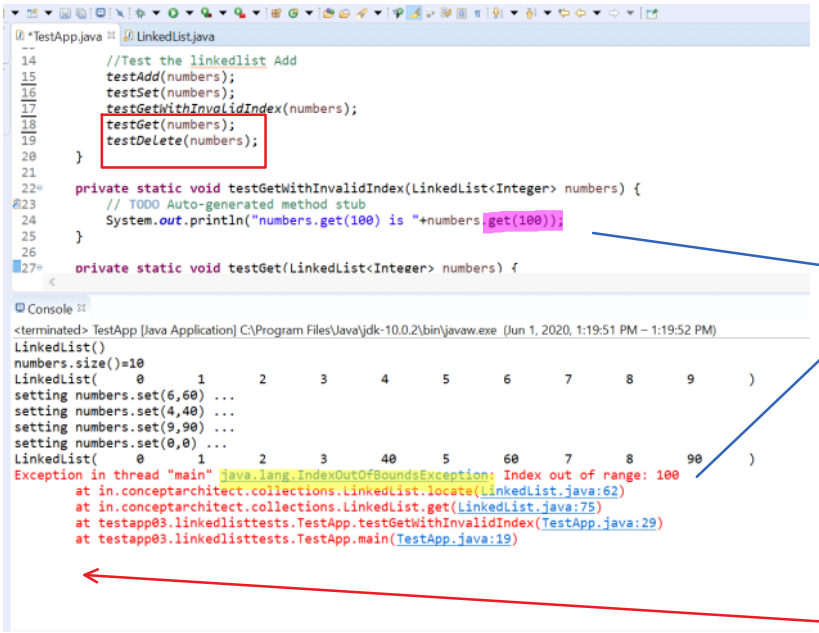
3. Test Result Changes when we change the order of the test.
   a. Are we sure getFunction is working currectly?
   b. which of the two gives test result more accurate?
   c. How to we decide the correct sequence?
   d. Will user of my code use the code in same sequence?
      i. Do they need to call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



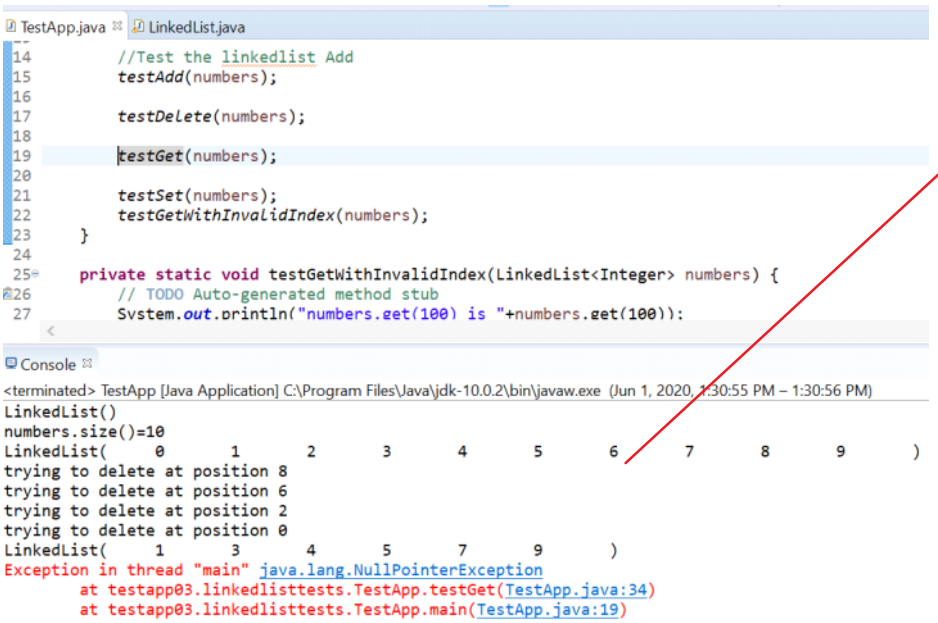
```
*TestApp.java  LinkedList.java
14          //Test the linkedlist Add
15          testAdd(numbers);
16          testSet(numbers);
17          testGetWithInvalidIndex(numbers);
18          testGet(numbers);
19          testDelete(numbers);
20      }
21
22      private static void testGetWithInvalidIndex(LinkedList<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) {

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          testSet(numbers);
22          testGetWithInvalidIndex(numbers);
23      }
24
25      private static void testGetWithInvalidIndex(LinkedList<Integer> numbers) {
26          // TODO Auto-generated method stub
27          System.out.println("numbers.get(100) is "+numbers.get(100));

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 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.