

What is BlueJ?

- BlueJ is a Java integrated development environment (IDE) which has been designed specifically for learning object oriented programming in Java.
- It is more convenient to use than the standard Java command line tools, and easier to learn than a full-featured IDE such as NetBeans or Eclipse.
- It will also help you understand how the classes in your object oriented programs are related to each other.

The screenshot shows the BlueJ IDE interface. On the left, a class hierarchy diagram displays the following structure:

```
graph TD; StackTester --> Stack; StackTester --> BracketChecker; StackTester --> Track;
```

On the right, the Object Inspector window shows the state of an object of type `stackTes2.stack_stack : Object[]`:

Index	Value
int length	5
[0]	"Test"
[1]	null
[2]	null
[3]	null
[4]	null



Stack Implementation in Java

- The Java Collections Framework includes a set of ready made data structure classes, including a Stack class.
- However, you will create your own stack class in order to learn how a stack is implemented.
- Your class will be a bit simpler than the Collections Framework one but it will do essentially the same job.
- A stack can be stored in: a static data structure OR a dynamic data structure
- Static data structures: These define collections of data which are fixed in size when the program is compiled.
- Dynamic data structures: These define collections of data which are variable in size and structure. They are created as the program executes, and grow and shrink to accommodate the data being stored.



The Stack Class

- This Stack class stores data in an array. The array reference type is **Object[]** which means that it can contain any kind of Java object. This is because of **polymorphism** – every Java class is a subclass of Object.
- The **constructor** creates a new array with its size specified as a parameter. The constructor does the job of the **Initialize** primitive described before.
- An instance variable **total** keeps track of how many items are stored in the stack. This changes when items are added or removed. The stack is full when total is the same as the size of the array.

Stack
- stack: Object []
- total: int
+ Stack(int)
+ push(Object) : boolean
+ pop() : Object
+ isEmpty() : boolean
+ isFull() : boolean
+ getItem(int) : Object
+ getTotal() : int



The Stack Class (2)

```
/**
 * Write a description of class Stack here.
 *
 * @author Imam M. Shofi
 * @version 1.0
 */
public class Stack
{
    private Object[] stack ;
    private int total; // to track number of items
    public Stack(int size) {
        stack = new Object[size]; // create array
        total = 0; // set number of items to zero
    }
}
```

+ push(Object) : boolean
+ pop() : Object
+ isEmpty() : boolean
+ isFull() : boolean
+ getItem(int) : Object
+ getTotal() : int



The Stack Class (3)

```
/**add an item to the array */
public boolean push(Object obj) {
    if ( isFull() == false) // checks if space in stack
    {
        stack[total] = obj; // add item
        total++; // increment item counter
        return true; // to indicate success
    }
    else {
        return false; // to indicate failure
    }
}
/** remove an item by obeying LIFO rule */
public Object pop() {
    if (isEmpty() == false) // check stack is not empty
    { // reduce counter by one
        Object obj = stack[total-1]; // get last item
        stack[total-1]= null; // remove item from array
        total--; // update total
        return obj; // return item
    }
    else {
        return null; // to indicate failure
    }
}
```



The Stack Class (4)

```

/** checks if array is empty */
public boolean isEmpty() {
    if (total ==0) {
        return true;
    }
    else {
        return false;
    }
}

/** checks if array is full */
public boolean isFull() {
    if (total ==stack.length) {
        return true;
    }
    else {
        return false;
    }
}

/** returns the item at index i */
public Object getItem(int i) {
    return stack[i-1]; // ith item at position i-1
}

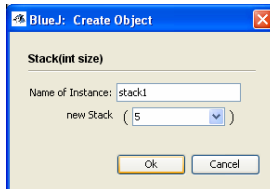
/** return the number of items in the array */
public int getTotal() {
    return total;
}

```



EXERCISE: Creating the Stack class

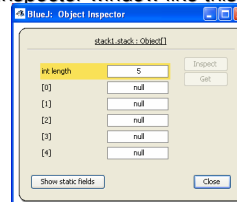
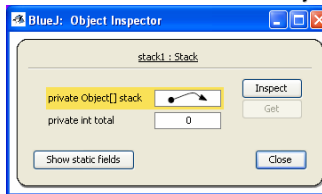
- Create a new BlueJ project called stacks and create a new class Stack using the aforementioned code.
- Create a new instance of Stack with size 5



- Inspect the Stack Instance.

What variable does it have?

- Click "OK" to select private Object[] stack and click the Inspect button. You should see an Object Inspector window like this:





Using a Stack

- Data structure classes are intended to be used in programs as utility classes which contain the data the program needs to work with. To use the Stack class, you need to know how to write code to call the Stack operations, for example to add data to the Stack.
- Remember that the Stack can hold any kind of data. The following test class shows how to use a Stack to hold Integer objects. Calls to Stack operations are shown in bold.

```
/** Class StackTester. */  
public class StackTester  
{  
    private Stack stack;  
    public StackTester(){  
        stack = new Stack(10);  
    }  
    public StackTester(Stack stack){  
        this.stack = stack;  
    }  
}
```

```
void pushNumber(int num)  
void popNumber()  
void checkIfEmpty()  
void checkIfFull()  
void listNumbersInStack()
```



Using a Stack (2)

```
/**push item into stack */  
public void pushNumber(int num) {  
    boolean ok = stack.push(new Integer(num));  
    if (!ok)  
        System.out.println("Push unsuccessful");  
    else  
        System.out.println("Push successful");  
}  
  
/**pop number from stack */  
public void popNumber() {  
    Integer result = (Integer) stack.pop();  
    if (result!=null)  
        System.out.println("Number is :" + result.intValue());  
    else  
        System.out.println("Pop unsuccessful");  
}  
  
/** check if stack is empty */  
public void checkIfEmpty() {  
    if (stack.isEmpty())  
        System.out.println("Stack empty");  
    else  
        System.out.println("Stack is not empty");  
}
```



Using a Stack (3)

```

/** check if stack is full */
public void checkIfFull() {
    if (stack.isFull())
        System.out.println("Stack full");
    else
        System.out.println("Stack is not full");
}

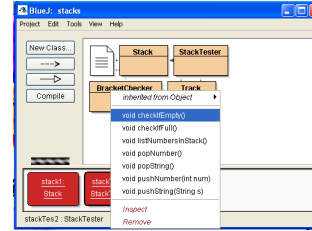
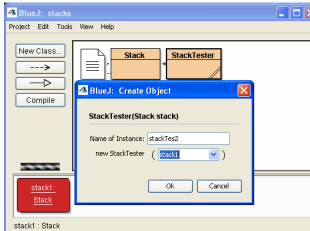
/** list the numbers in stack */
public void listNumbersInStack() {
    if (stack.isEmpty()) {
        System.out.println("Stack empty");
    }
    else {
        System.out.println("Numbers in stack are: ");
        System.out.println();
        for (int i=stack.getTotal(); i>=1; i--) {
            System.out.println(stack.getItem(i));
        }
        System.out.println();
    }
}

```



Exercise: Using a Stack

- Add a new class StackTester to your stacks project using the above code.
- Create a new instance of Stack with size 5.
- Create a new instance of StackTester and select your Stack instance in the object bench as the parameter in the constructor. This means that you will be testing the Stack you created in the previous step.
- Call the checkIfEmpty method of your StackTester. **What was the result?**



- Call the pushNumber method of your StackTester to add the number 5 to the stack. Inspect the Stack instance. Repeat this to add the number 7 and repeat again to add the number 2.
What result would you expect if you pop from the Stack?
- Call the popNumber method of your StackTester and check that you got the correct result.
- Call the methods of StackTester as needed to add and remove more numbers and
- check what happens when the Stack is full and when it is empty.



For Your EXERCISE: Storing other types of data

- Modify the StackTester class to store String objects in a Stack instead of Integer objects, and test in a similar way to the above.
- You should not have to change the Stack class at all



EXERCISE: A practical application of the Stack class

- Compilers make use of stack structures. This example shows a simple illustration of the way a stack can be used to check the syntax of a program. It uses the Stack class you have created.
- In the example, a Stack is used to check that the braces { and } used to mark out code blocks in a Java source file are matched – i.e. that there are the same number of { as }
 - The source file is read character by character.
 - Each time a { is read an object (any object, it doesn't matter what) is pushed into the stack.
 - Each time a } is read an object is popped from the stack.
 - If the stack is empty when a } is read then there must be a missing { .
 - If the stack is not empty at the end of the file then there must be a missing }



EXERCISE: A practical application (the code)

```
import java.io.*;
import java.net.URL;
/** class BracketChecker. */
public class BracketChecker {
    private Stack myStack = new Stack(100);
    public void check() throws IOException {
        // opens file in project directory
        URL url = getClass().getClassLoader().
            getResource("Track.java");
        if (url == null)
            throw new IOException("File not found");
        InputStream in = url.openStream();
        int i;
        char c;
        // read each character in the file. add a new object to the stack every time an opening brace is
        // read. remove an object every time a closing brace is read
        while ((i = in.read()) != -1) {
            c = (char)i;
            System.out.print(c);
            // if character is closing brace, the stack should not be empty
            if (c == '}') {
                if (myStack.isEmpty()) System.out.println("\n***** Error: missing { *****");
                // remove top object of stack
                else myStack.pop();
            }
            else {
                if (c == '{')
                    myStack.push(new Object());
            }
        }
        // stack should end up empty if braces balance
        if (!myStack.isEmpty()) System.out.println("\n *****Error: missing } *****");
        in.close();
    }
}
```



For Your Home/own EXERCISE: Algorithms using stacks

A stack is a natural structure for reversing a list as it will output values in the reverse order in which they are stored.

- a) Design an algorithm, using a stack, to read five characters from a keyboard and display them in reverse order.
- b) Design an algorithm, using a stack, which will convert a decimal number to binary.
- c) Design an algorithm, using a stack, which will test whether an input string is palindrome or not.

A palindrome is a word / sentence which reads the same backwards as forward.