Contents(1)

- **Goal and Scope**
- **Basic Concepts on OOT**
  - Basic Concepts to represent the world
  - Basic Concepts for Reuse
  - Information Hiding Principle and Java Program
  - Superiority of OOT
- **Modeling Techniques**
  - Static Model: Class and Association
  - Dynamic Model: State Machine
  - Dynamic Model: Interaction Diagram
  - Concurrency Description: Active Object and Multi-thread Programming
  - Outline of UML2.0

Java Programming and Information Hiding Principle

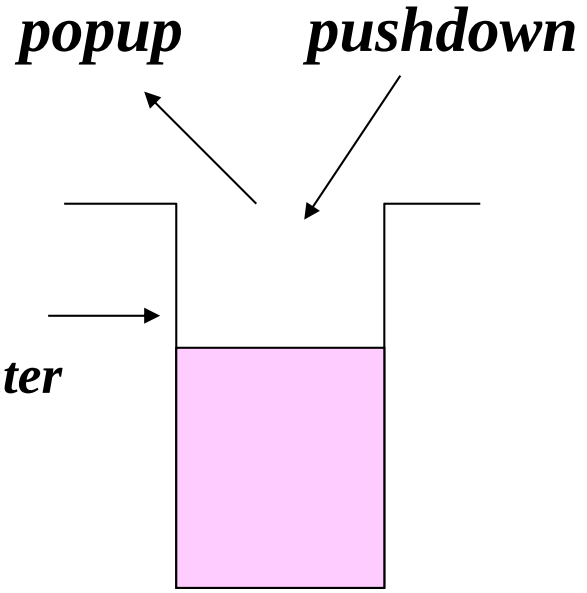
JAIST Koichiro Ochimizu

Example (Stack)

- Last-in First-out
- push-down, pop-up

Operations specific to stack

```
void push( String str ) {
   stack[ stack_p++ ] = new String( str );
}
String  pop(  ) {
   return new String( stack[ --stack_p] );
}
boolean empty(  ) {
   return ( stack_p = = 0);
}
 String  val(  ) {
   return new String( stack[ stack_p - 1] );
}
```



Structure of Object oriented program

- An object oriented program consists of classes
- public class balanced { // balanced.java
- . . .
- }
- public class Stack { // Stack.java
- . . .
- }

Structure of a class definition

- A class consists of several local variables and methods
- public class Stack {
- int max = 10; //capacity of stack
- int stack_p = 0; //stack pointer
- String[] stack; //implementation of stack by an array
- Stack( ) { stack = new String[max];}
- //instanciation of object “stack”
- void push(String str) {}
- String pop( ) {}
- boolean empty( ) {}
- boolean full( ) {}
- String val( ) {}
- }

Instantiation

- Instantiate an object based on class definition

- **Stack stack1 = new Stack( );**

Set and refer the state of the object

- We can create multiple objects based on the same class definition, if necessary.
- **Stack stack1 = new Stack( );**

  Stack stack2 = new Stack( );

- The structure of stack1 is same to the structure of stack2.
- As computation proceed, the state of the object stack1 is usually different from the state of the object stack2, because the method Invocation Is performed to each object.
- **stack1.pop( );**

Example

```
public class balanced{
  public balanced( ) { }  //constructor

  public static void main( String[ ] args ) // java balanced  "(()())"
    String in = new String( args[0]); //translate input to String object
    System.out.println( in );
    balanced b = new balanced(); // instanciation of instance b
    b.parse( in );
  }

  public void parse( String input ) { // beginning of method parse
    int check = 0;  // definition of local variable check and its initialization
    char ch; // definition of local variable ch which contains next character
    Stack stack1 = new Stack( );
    // instanciation of object stack1 based on class definition of  class Stack
```

Example

```
    for( int  i = 0; i < input.length(); i++) {
      ch = input.charAt(i);
      if( ch == '(' ) {// If the next character Is (, then pushdown it.
        if( !stack1.full() ){ stack1.push(  "(" ); check++;}
        else { System.out.println("Stack overflow."); error(); }
     }else if( ch = ')' ) { //If the next character is ), popup ( from stack
        if( !stack1.empty() ){ stack1.pop(  ); check--;}
        else {System.out.println("Stack underflow."); error(); }
     }else { // The next character Is neither ( nor ).
        System.out.println("X."); error()
      }
    }
    if( stack1.empty()) System.out.prinln("done") else error();
  }
  void error( ) {System.out.prinln("Error"); System.exit(0);}
}
```

Example

```
public class Stack{
    int max = 10; //capacity of the stack
    int stack_p = 0; //stack pointer which Indicates next open place
    String[ ] stack; // Implementation of stack by an array

    Stack( ) { stack = new String[max];}
   // Instanciation of stack object with capacity m. Element type is string array

    void push( String str ) {  stack[ stack_p++ ] = new String( str ); }
    String  pop(  ) {  return new String( stack[ --stack_p] ); }
    boolean empty(  ) {  return ( stack_p = = 0);}
    boolean  full(  ) {  return ( stack_p = = max );}
    String  val(  ) { return new String( stack[ stack_p - 1] ); }
}
```

Information Hiding Principle

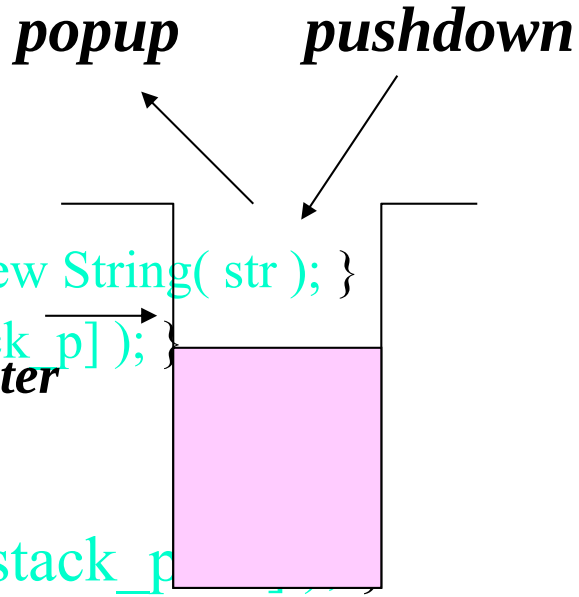
```
public class Stack{
    int max = 10; //size of stack
    int stack_p = 0; // stack pointer, indicate next position
    String[ ] stack; // implement stack by array

    Stack( ) { stack = new String[max];  }
    // instanciate an instance of stack with size
    void push( String str ) {  stack[ stack_p++ ] = new String( str ); }
    String  pop( ) {  return new String( stack[ --stack_p] ); }
    boolean empty( ) {  return ( stack_p = = 0);  }
    boolean  full( ) {  return ( stack_p = = max );  }
    String  val( ) { return new String( stack[ stack_p
}
```

can not refer from outside

can refer from method

can refer from outside



Let's execute a Java program

- Compile
  - % javac file-name
  - File name should be class-name.java
  - Executable code of java is xxx.class
- Execution
  - java xxx

Exercise

- Change the implementation of stack by array to by linked list. Check the range of modification when you change the data structure from array to linked list (Information Hiding)