CS 251
Intermediate Programming
Postfix Calculator Code

Brooke Chenoweth

University of New Mexico

Fall 2026

Stack Interface

```
public interface Stack<T> {
  /** Is the stack empty? */
  boolean isEmpty();

  /** Push item onto top of stack. */
  void push(T val);

  /** Pop item from top of stack and return it. */
  T pop();

  /** Return top item on stack without removing it. */
  T peek();
}
```

DoubleStack – Using LinkedList

```
import java.util.*;

public class DoubleStack implements Stack<Double> {
  private Deque<Double> values = new LinkedList<>();

  public boolean isEmpty() {
    return values.isEmpty();
  }

  public void push(Double val) {
    values.push(val);
  }

  public Double pop() {
    return values.pop();
  }

  public Double peek() {
    return values.peek();
  }
}
```

DoubleStack – Using ArrayList

```
import java.util.*;

public class DoubleStack implements Stack<Double> {
  private List<Double> values = new ArrayList<>();

  public boolean isEmpty() {
    return values.isEmpty();
  }

  public void push(Double val) {
    values.add(val);
  }

  public Double pop() {
    return values.remove(values.size() - 1);
  }

  public Double peek() {
    return values.get(values.size() - 1);
  }
}
```

DoubleStack – Extending LinkedList

```
import java.util.*;

public class DoubleStack
      extends LinkedList<Double>
      implements Stack<Double> {
}
```

This one is a bit of a stunt. LinkedList just happens to have methods that match the Stack interface.

Operator Interface

```
import java.util.*;

public interface Operator {

  /**
   * How many arguments does this operator take?
   * @return number of arguments required by operator.
   */
  int numArgs();

  /**
   * Evaluate this operator using the given arguments.
   * Should not modify the list of arguments.
   * @param args Argument list.
   * @return Result of the operation.
   */
  double eval(List<Double> args);
}
```

PostFixCalculator – Member Variables

```
import java.util.*;

public class PostFixCalculator {
  /** Stack of operands */
  private Stack<Double> stack = new DoubleStack();

  /** Map of operator strings to operator objects. */
  private Map<String, Operator> opMap = new HashMap<>();

  // parent class for all 2-arg operators
  private static abstract class BinaryOperator
                                  implements Operator {
    public int numArgs() { return 2; }
    public double eval(List<Double> args) {
      return eval(args.get(0), args.get(1));
    }
    public abstract double eval(double n1, double n2);
  }
```

PostFixCalculator – Constructor (+,-)

```
public PostFixCalculator() {
  Operator add = new BinaryOperator() {
    public double eval(double n1, double n2) {
      return n1 + n2;
    }
  };
  opMap.put("+", add);
  opMap.put("add", add);

  Operator sub = new BinaryOperator() {
    public double eval(double n1, double n2) {
      return n1 - n2;
    }
  };
  opMap.put("-", sub);
  opMap.put("sub", sub);
```

PostFixCalculator – Constructor (*,/)

```
Operator mult = new BinaryOperator() {
  public double eval(double n1, double n2) {
    return n1 * n2;
  }
};
opMap.put("*", mult);
opMap.put("mult", mult);

Operator div = new BinaryOperator() {
  public double eval(double n1, double n2) {
    return n1 / n2;
  }
};
opMap.put("/", div);
opMap.put("div", div);
```

PostFixCalculator – Constructor (Print)

```
    Operator print = new Operator() {
      public int numArgs() {
        return 1;
      }
      public double eval(List<Double> args) {
        System.out.println(args.get(0));
        return args.get(0);
      }
    };
    opMap.put("=", print);
    opMap.put("print", print);
  }
```

PostFixCalculator – Constructor (Sqrt)

```
    Operator sqrt = new Operator() {
      public int numArgs() {
        return 1;
      }
      public double eval(List<Double> args) {
        return Math.sqrt(args.get(0));
      }
    };
    opMap.put("sqrt", sqrt);
  }
```

PostFixCalculator – storeOperand

```
    public void storeOperand(double operand) {
        stack.push(operand);
    }
}
```

PostFixCalculator – evalOperator

```
public void evalOperator(String opStr) {
  // Look up operator
  Operator op = opMap.get(opStr);

  // Pop operands off stack
  List<Double> args = new ArrayList<Double>();
  for(int i = op.numArgs(); i > 0; --i) {
    args.add(0, stack.pop());
  }

  // Evaluate operator, push result
  stack.push(op.eval(args));
}
```