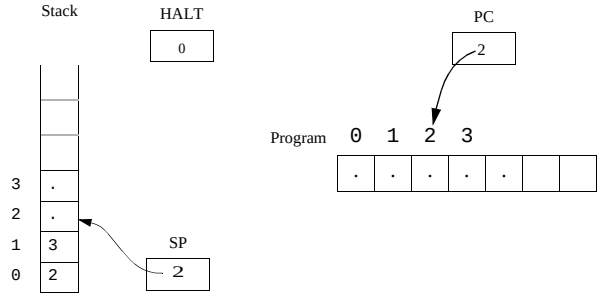
SaM

Implementing a Stack Machine

An Example of Program Design using Inheritance and Interfaces

Stack Machine Components

- *Stack* of values
- *SP* (Stack Pointer)
- *Program* (Array of *Instructions*)
- *PC* (Program Counter)



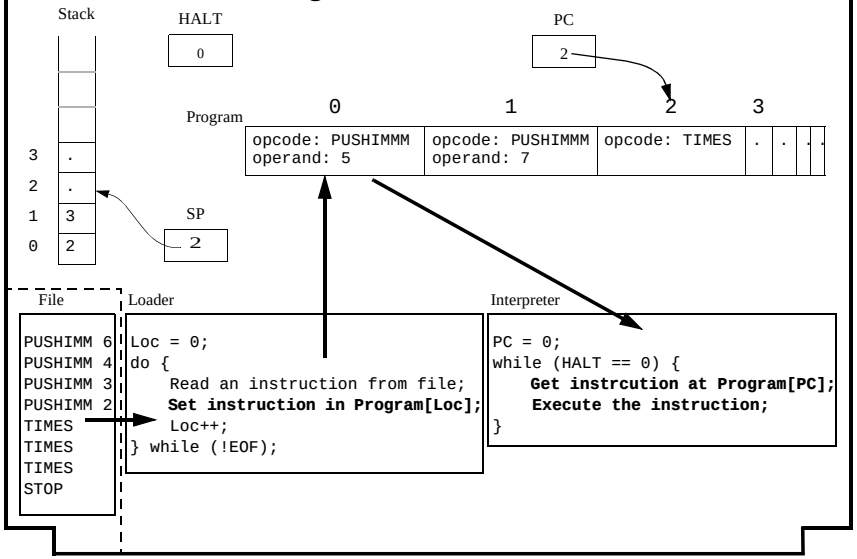
SaM Program

Example:

```
PUSHIMM 6
PUSHIMM 4
PUSHIMM 3
PUSHIMM 2
TIMES
TIMES
TIMES
STOP   // result 120 on top of stack
```

- How is a SaM program made available to the stack machine for execution?

Program Execution in SaM



Interface for Stack Machines

```
interface SaM {
    //getter and setter methods
    void setPC(int v);
    int  getPC();
    void setSP(int v);
    int  getSP();
    void setStackElement(int a, int v); //Stack[a] = v;
    int  getStackElement(int a);        //return Stack[a]
    // ...
    ? getInstruction(int a);              // instruction at Program[i]
    boolean setInstruction(int a, ? ins); // set instruction at Program[i]
    // ...
    void run( );                   // the main xqt loop
    int load(String s);                 // loader (of the program)
    //convenient setter and getter methods
    void push(int v);
    int  pop();
    void incPC();
    void halt();
}
```


Implementing Instruction Execution

- Switching on `OPCODE` of the Instruction in the execution loop

```
int OPCODE = getOpcode(Program[PC]);
switch (OPCODE) {
    case PUSHIMM: doPUSHIMM(); break;
    case TIMES:   doTIMES(); break;
    ...
}
```

Major changes in the code to incorporate new instructions!

The relevant code for instruction execution is spread in various methods.

Better solution: Use Inheritance

Instruction Interface

- Note that each instruction can execute itself on a stack machine that implements the `SaM` interface.

```
interface I_Instruction {
    void readOperand(CS211InInterface inf);
    void execute(SaM machine);
    void translate(SymbolTable ST, SaM m);
}

interface SaM {
    // ...
    I_Instruction getInstruction(int a);
    boolean setInstruction(int a, I_Instruction ins);
    // ...

}
```


Instruction Execution

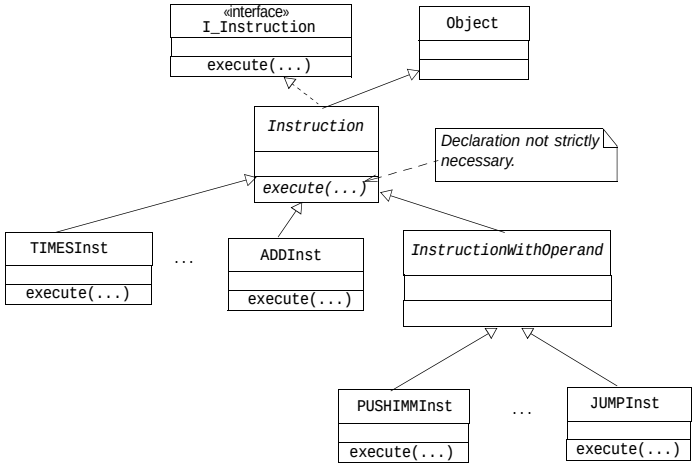
- How would a concrete class implement the method `execute()` in `I_Instruction`.

Each instruction should know how to should execute itself.

```
class Instruction implements I_Instruction {
    /...
    public void execute(SaM machine) {
         int OPCODE = getOpcode();
         switch (OPCODE) {
             case PUSHIMM: doPUSHIMM(); break;
             case TIMES:   doTIMES(); break;
             ...
         }
    }
}
```

- Not much better than before!

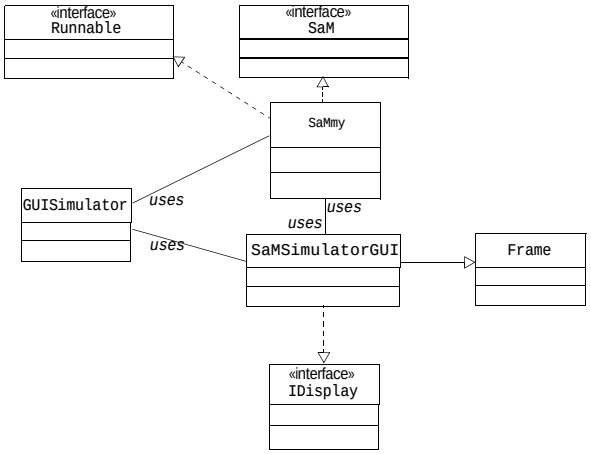
Concrete Instructions



Implementing Instructions

```
interface I_Instruction {
    void execute(SaM machine);
    //...
}
abstract class Instruction implements I_Instruction {
    abstract public void execute(SaM machine);  // Strictly not necessary.
    //...
}
class ADDInst extends Instruction {
    public void execute(SaM machine) {...}
    //...
}
abstract class InstructionWithOperand extends Instruction {
   // Does not implement execute(SaM machine)
   //...
}
class PUSHOFFInst extends InstructionWithOperand {
    public void execute(SaM machine) {...}
    //...
}
```


Concrete Stack Machine



Implementing Stack Machine

```
public class GUISimulator {
    public static void main(String[] args) {
       SaM Sumatra = new SaMmy(); //My machine is called Sumatra
        IDisplay g = new SaMSimulatorGUI(Sumatra);//open GUI
        Sumatra.setGUI(g);
    }
}

public class SaMmy implements SaM, Runnable {
    // ...
}

public class SaMSimulatorGUI extends Frame implements IDisplay {
    // ...
}
```