Data Structures and Object-Oriented Design II

Spring 2014

Carola Wenk

Stacks

- Are the methods in this class guaranteed to work? What kind of specifications can we guarantee to ensure the correctness of push and pop?
- How does pop handle empty stacks?

```
public int pop() {
   return S[top--];
}
```

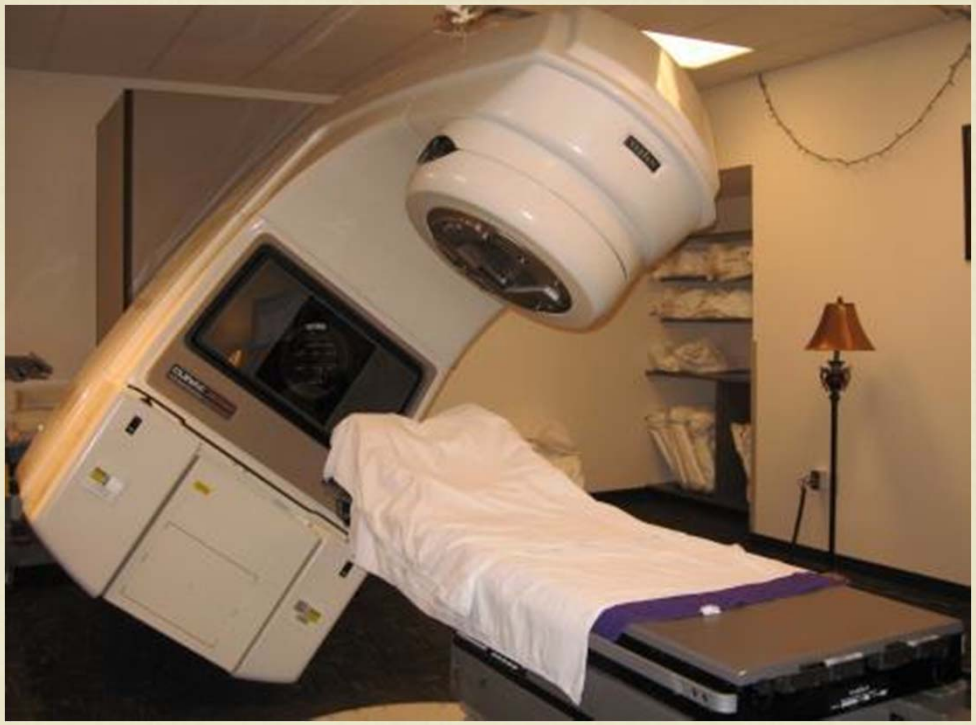
```
public int pop() {
  if (top >= 0)
    return S[top--];
}
```

```
public int pop() {
  if (top >= 0)
    return S[top--];
  else
    return -999;
}
```

```
public int pop() {
  if (top >= 0)
    return S[top--];
  else
    throw new RuntimeException("Stack is empty");
}
```

Software Can Kill

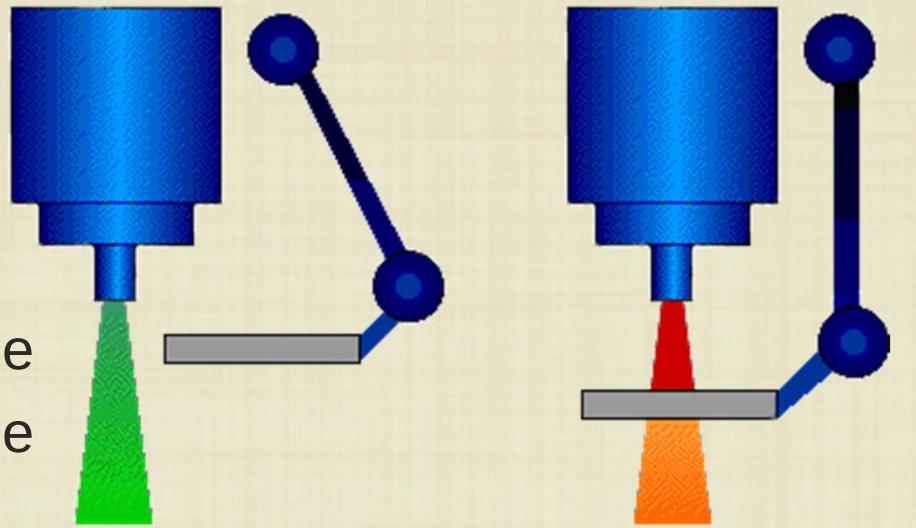
- Meeting specifications is of critical importance when software is used to control dangerous hardware.



The Therac-25 relied on a software system to deliver different kinds of radiation: electron beam therapy, and X-ray therapy. These two types of radiation are used to treat different types of cancer.

Software Can Kill

Meeting specifications is of critical importance when software is used to control dangerous hardware.



$B =$ False
$S =$ False

Electron Mode
$M =$ True

X-Ray Mode
$M =$ False

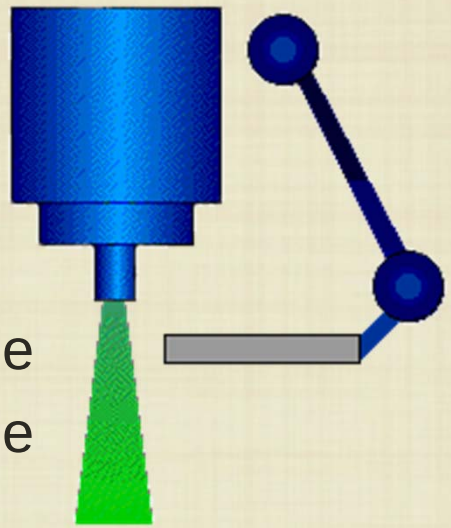
Specification

$$(M \rightarrow \neg B \wedge \neg S) \wedge$$
$$(\neg M \rightarrow B \wedge S)$$

The Therac-25 radiation therapy system needed to guarantee that a shield is always in place when in “X-ray” mode. In certain instances the specification was not met, and as a result patients received 100x the allowable amount of radiation.

Software Can Kill

Meeting specifications is of critical importance when software is used to control dangerous hardware.

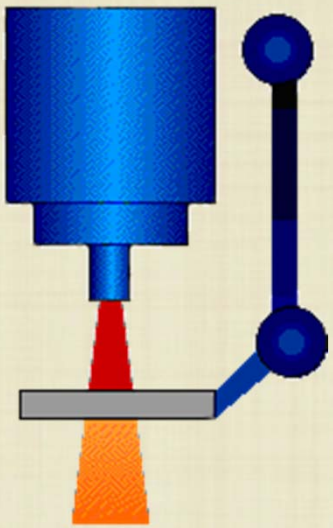


$B =$ False

$S =$ False

Electron Mode

$M =$ True



X-Ray Mode

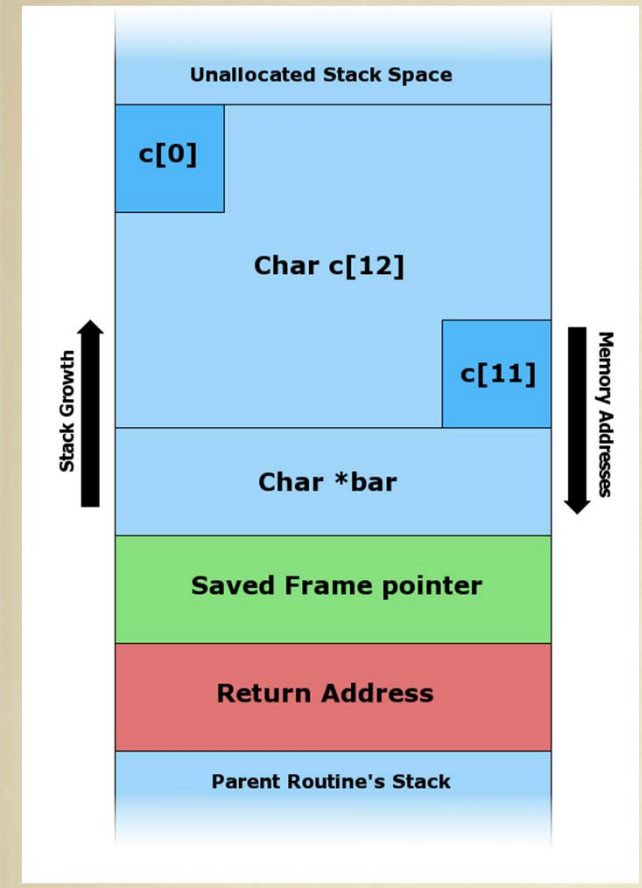
$M =$ False

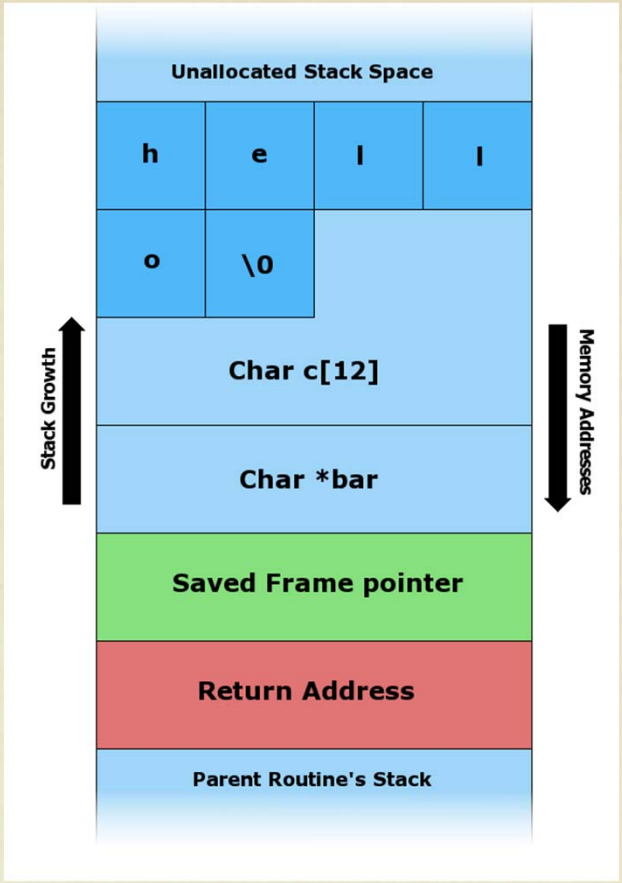
Specification

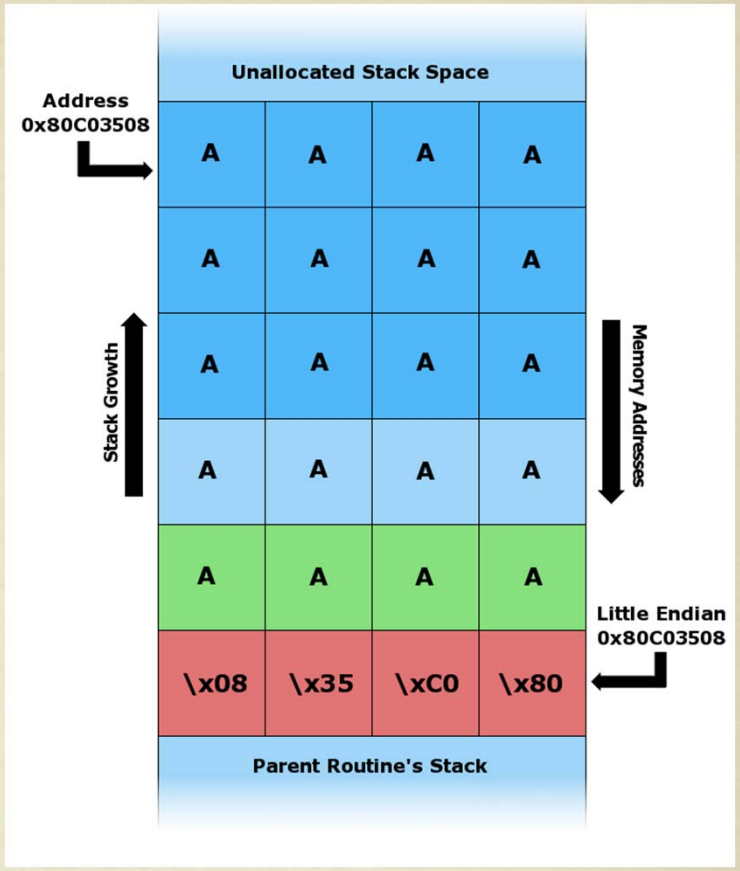
$$(M \rightarrow \neg B \wedge \neg S) \wedge$$
$$(\neg M \rightarrow B \wedge S)$$

An investigative commission found that the Therac-25 failed due to poor software development practices that led to a system that was difficult to verify or test.

Stack “Buffer Exploits”





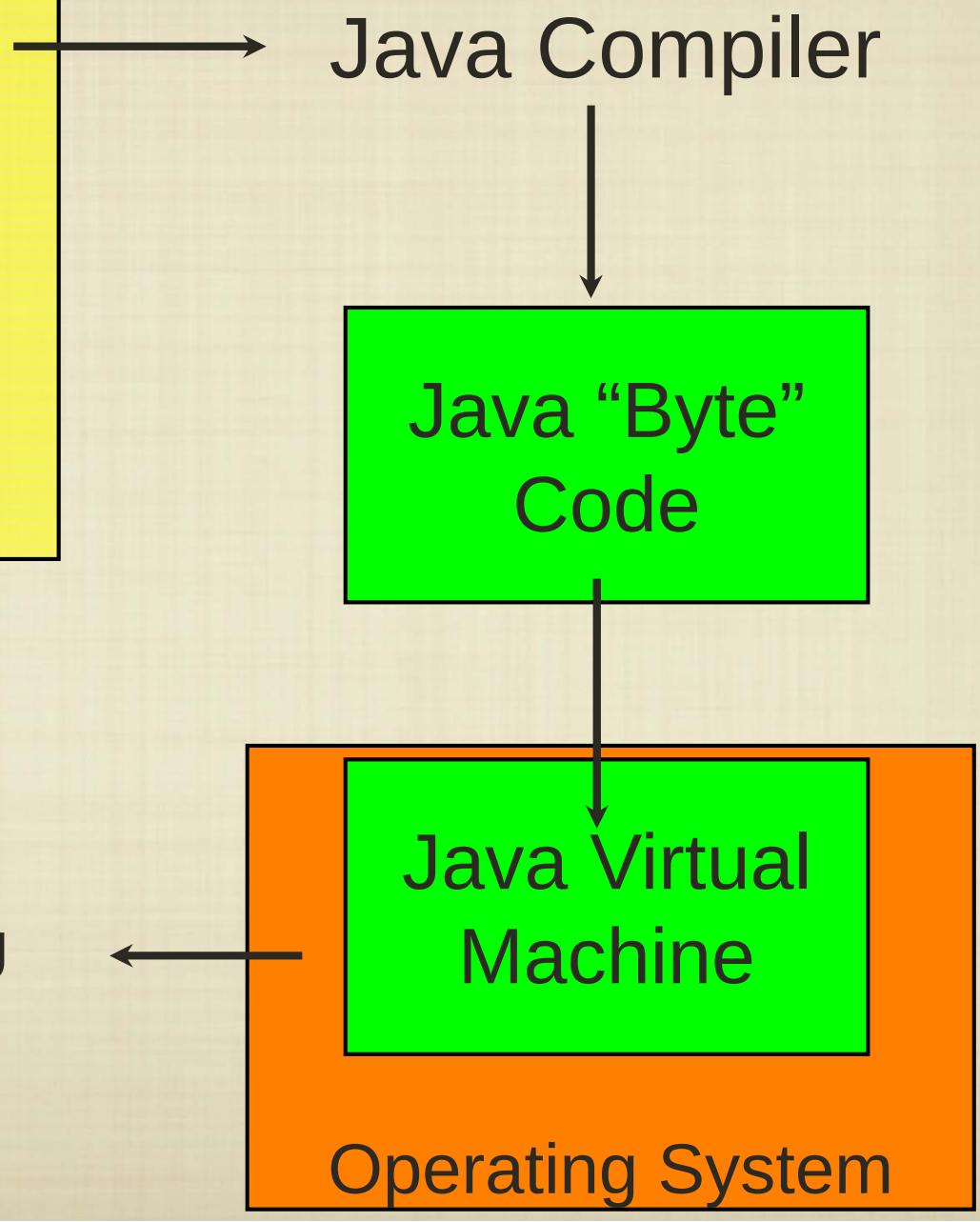


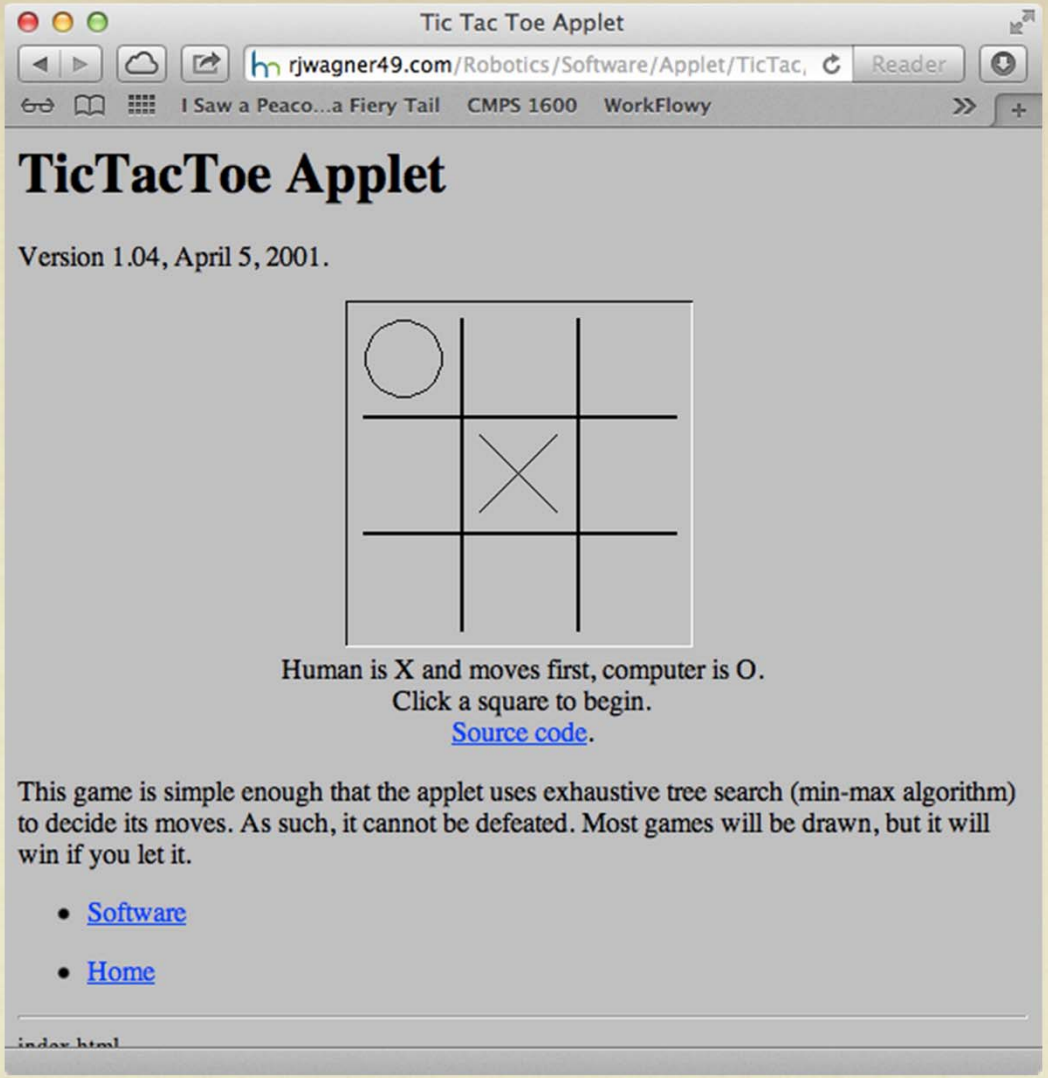
[wikipedia]

Nearly every operating system utilizes a stack to manage the function calls. Programs can exploit the lack of a stack buffer check to modify the operating system and execute arbitrary code!

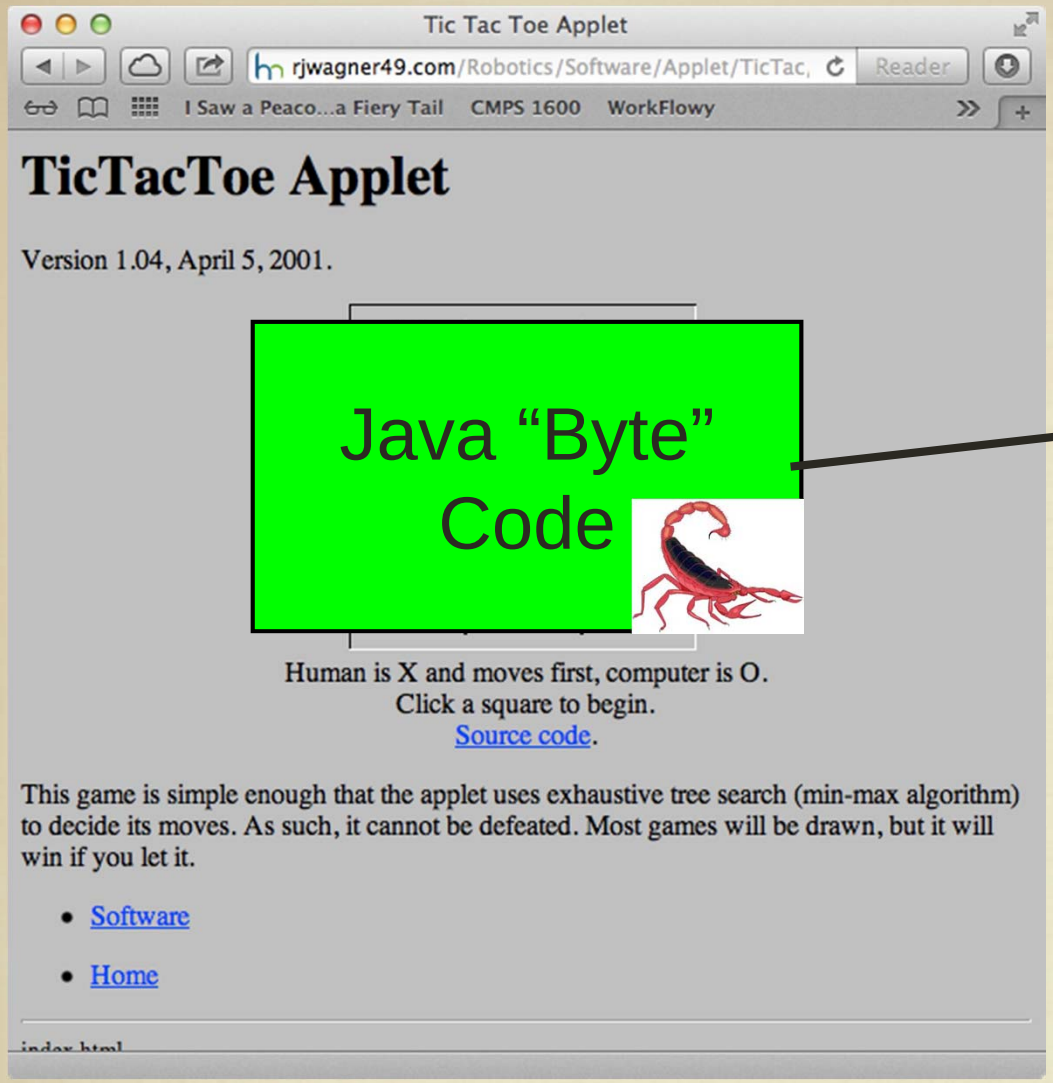
Java Runtime System

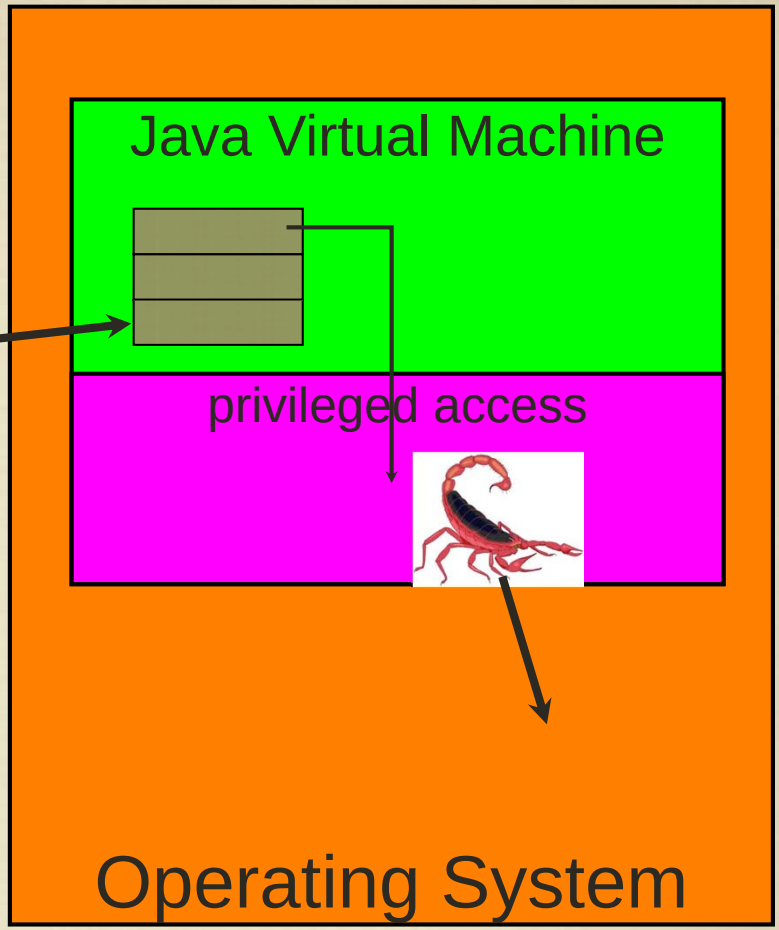
```
import --;

class HelloWorld {
public void f(int x1, char x2, ...) {
...
}

public long g(boolean y1, float y2, ...) {
...
}

private int h(double z1, int z2, ...) {
...
}

public static void main(String [] args) {
       System.out.println(“hello world!”)
       System.out.println(“goodbye world!”)
}
}
```





Recent “zero-day” exploits (compromising Facebook, Twitter, Apple) utilize Java applets to circumvent OS security and install malware.

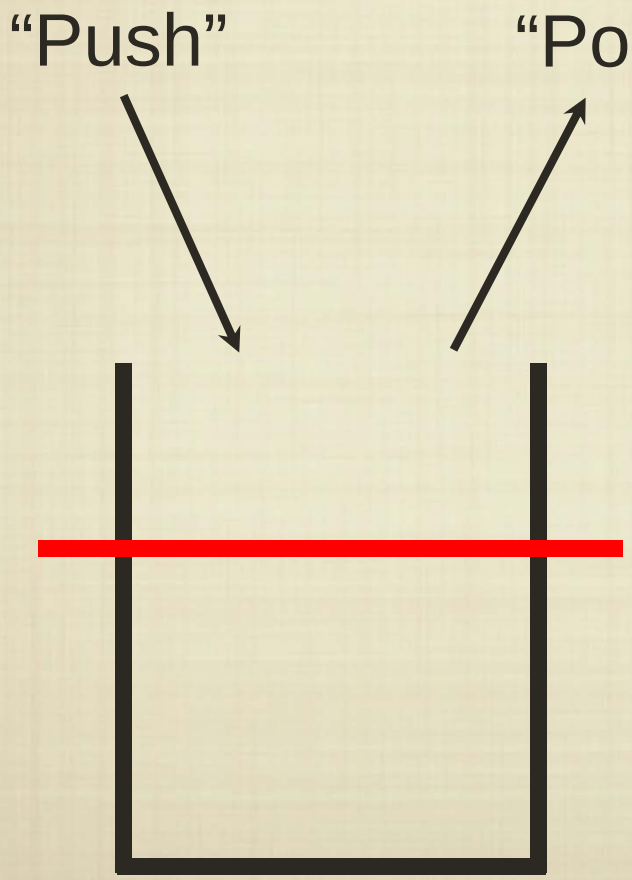




Recent "zero-day" exploits involving Java hide malicious code in Java applets that can circumvent built-in security provisions to install malware.

Limitations of Arrays

- One limitation of our implementation is that we have a set capacity for our storage.




```
class Stack {

...

public Stack(..) {
      ...
}

public int pop() {
      ...
}

public void push(int x) {
      ...
}
```

Stacks

How do we remove this limitation? In Python, we developed linked data structures that could “declare” new storage. What about in Java?

```
class Node:
  def __init__(self, data = None, next = None):
    self.data = data
    self.next = None

  def __str__(self):
       return str(self.data)

L = Node(‘a’)
L.next = Node(123)
```

Stacks

How do we remove this limitation? In Python, we developed linked data structures that could “declare” new storage. What about in Java?

```
class Node{
  private int data;
  private Node next;

  public Node(int d){
    data = d;
    next = null;
  }

  public String toString() {
     return Integer.toString(data);
  }
}
```

```
L = new Node(1);
L.next = new Node(123);
```

Array-Based Stack vs. DynamicStack

```
public class ArrayStack {
  final static int DEFAULT_CAPACITY=50;
  private int[] S;
  private int top;

  public ArrayStack(){
    this(DEFAULT_CAPACITY);
  }

  public ArrayStack(int capacity){
    S = new int[capacity];
    top=-1;
  }

  public void push(int x){
    S[++top]=x;
  }

  public int pop(){
    if(top>=0)
      return S[top--];
    else
      throw new RuntimeException("Stack is empty.");
  }
}
```

```
public class Tester{
  public static void main(String[] args) {
    ArrayStack stack = new ArrayStack();
    stack.push(5);
    System.out.println("popped: "+stack.pop());
  }
}
```

```
public class DynamicStack implements Stack{

  private class StackNode {
    private int data;
    private StackNode next;

    public StackNode(int d){
      data = d;
      next=null;
    }
  }

  private StackNode top = null;

  public void push(int x) {
    StackNode temp = new StackNode(x);
    temp.next = top;
    top = temp;
  }

  public int pop(){
    if (top == null)
      throw new RuntimeException("Stack empty!");

    int x = top.data;
    top = top.next;
    return x;
  }
}
```

If we change `ArrayStack` to `DynamicStack`, the code still works.

Java Interfaces

- We can specify that a Java class implements a particular kind of functionality defined as an `interface`.

```
public interface Stack {
  public int pop();
  public void push(int x);
}
```

```
public class ArrayStack implements Stack {

  ...

}
```

```
public class DynamicStack implements Stack {

  ...

}
```

```
public class Tester {
  public static void main(String[] args) {
    Stack stack = new ArrayStack();
    stack.push(5);
    System.out.println("popped: "+stack.pop());
  }
}
```

If we change `ArrayStack` to `DynamicStack`, the code still works.