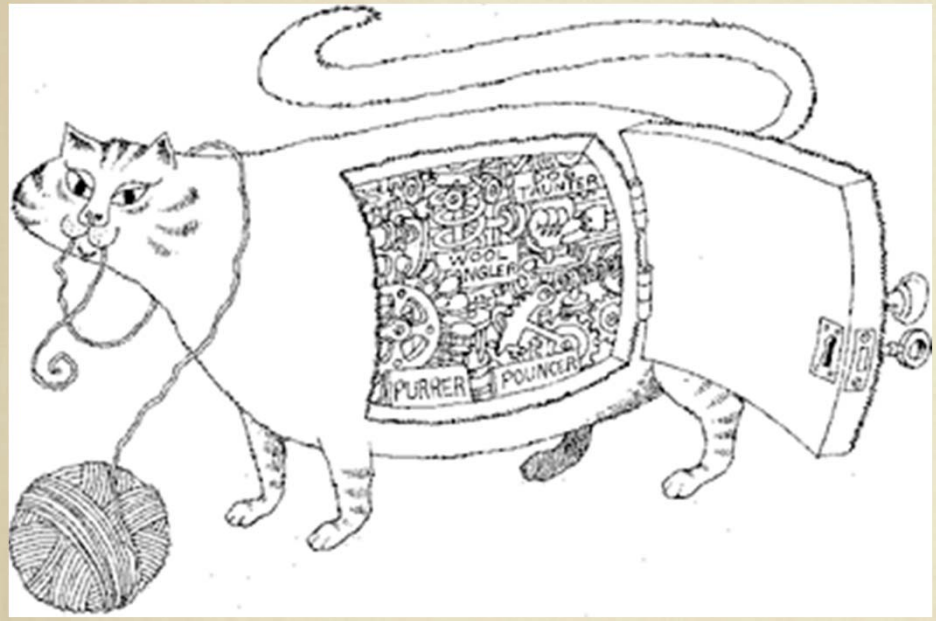
Data Structures and Object-Oriented Design

Spring 2014

Carola Wenk

Data Types So Far

- With classes we can define our own ‘abstract data types’. The purpose of classes is to package data and functionality.

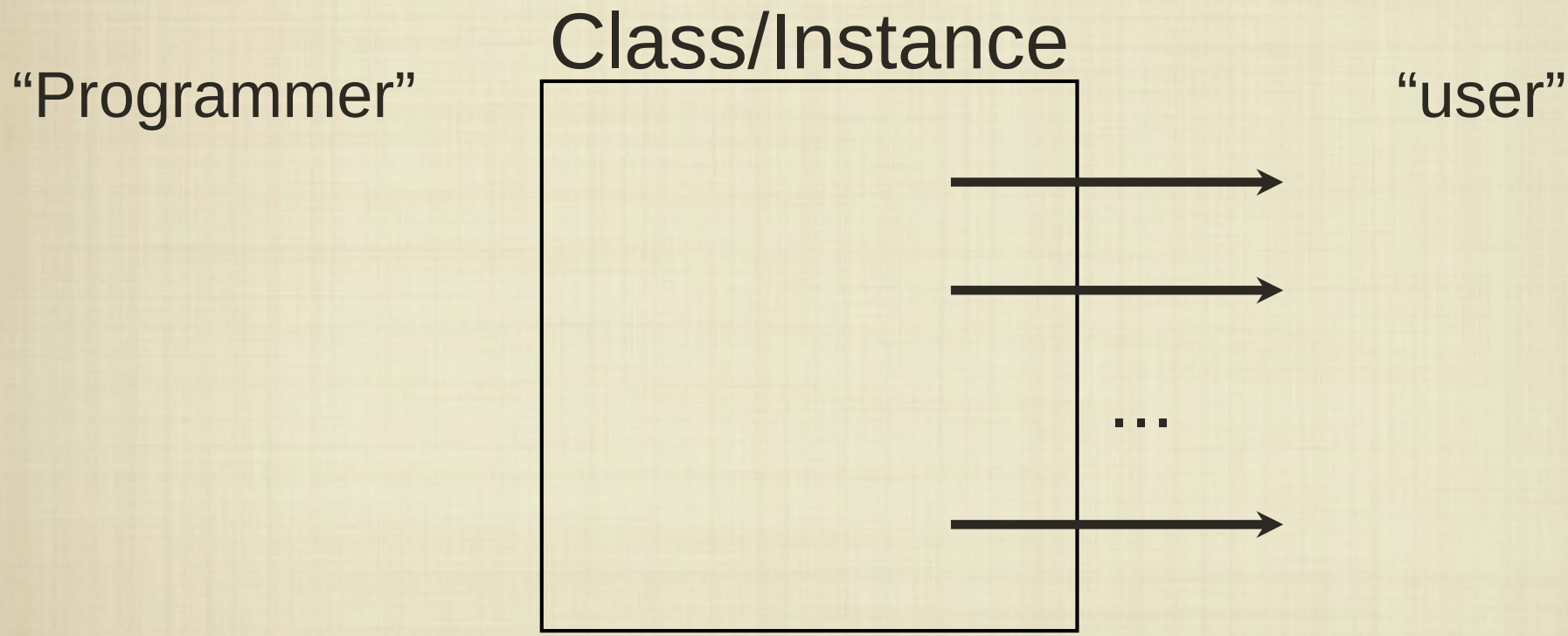


[Booch ’98]

The goal in packaging is to hide complexity and only expose data/functionality necessary.

Data Types So Far

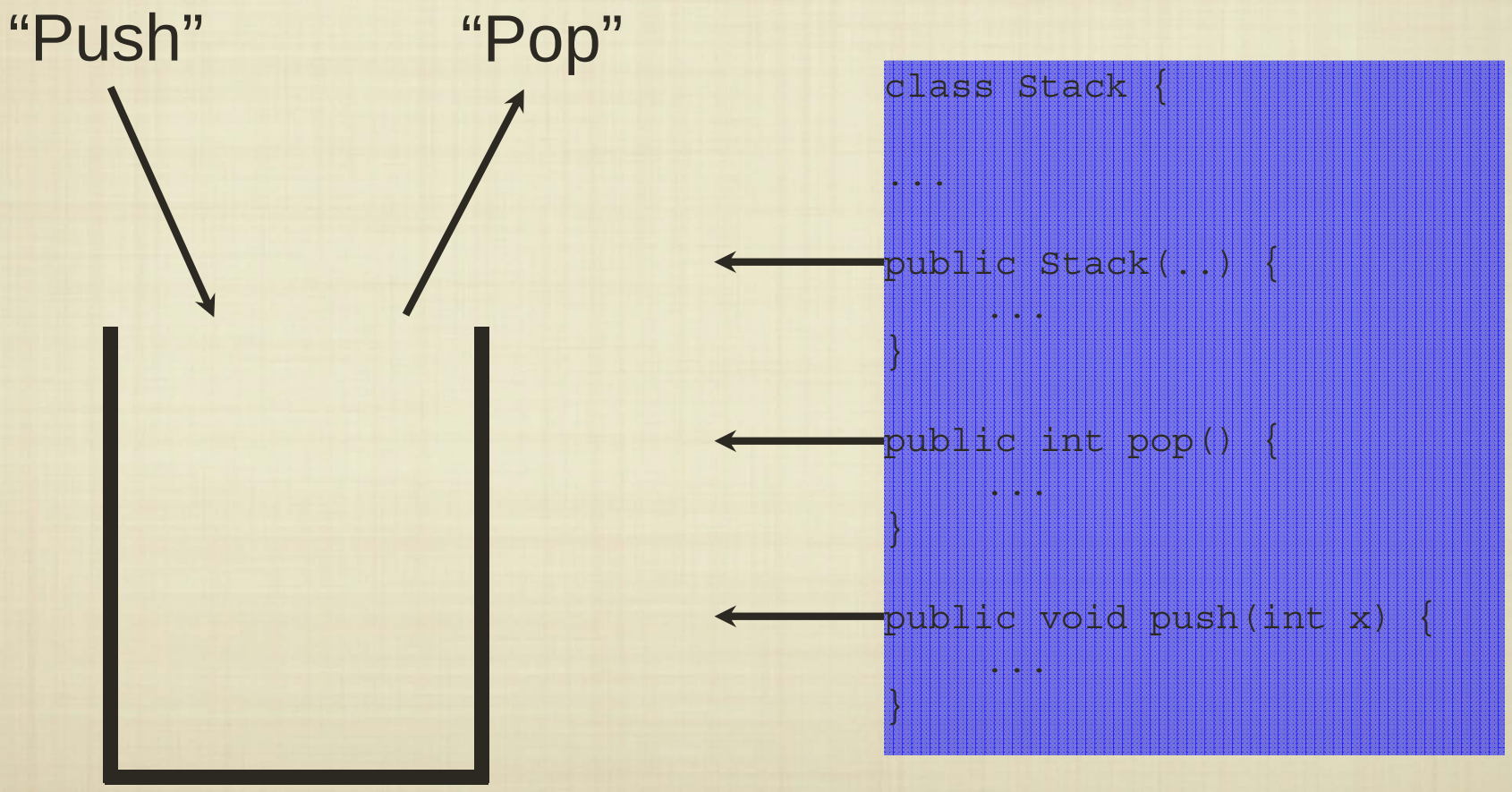
- With classes we can define our own ‘abstract data types’. The purpose of classes is to package data and functionality.



The programmer provides all the details, the user just invokes what is necessary and expects correct behavior.

Stacks

- A stack is a “last-in, first-out” data structure. The functionality is very simple, but how do we implement it?



Stacks

A stack is a “last-in, first-out” data structure. The functionality is very simple, but how do we implement it?

```
class Stack {

...

public Stack(...) {
    ...
}

public int pop() {
    ...
}

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

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--];
}
```

Does not run.

Java Exceptions

- In Python, we were allowed to write functions like this:

```
def f(a, b):
  if (a + b > 0):
    return a+b
```

Note that this kind of function can return nothing at all, even though we can use it in an assignment statement or comparison.

This is not allowed in Java, and so we really need a way to avoid returning something of the declared type when we “need” to.

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");
}
```

Java Exceptions

- Java allows the programmer to subvert the type system to return error messages:

```
...

try { f(a, b); }
catch(Exception e) {
System.out.println(e.getMessage());
}
finally {
// “clean up” area
}
...
```

```
...

public int f(int a, int b) throws
Exception {
if (a + b > 0)
     return a+b;
else
     throw new Exception(“Error!”);


...
```

Some exceptions must always be handled (either by the caller or higher up), while others are handled by the system.

Java exceptions are just classes, and can be treated as such.

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");
}
```