Sounds simple, right?

Here is a **Java** method that we would like to write equivalently in C++:

```java
static String duplicate(String in) {
    return in + in;
}

String s = "hello";
System.out.println(duplicate(s));  // hellohello
System.out.println(s);            // hello
```

© 2011 Fawzi Emad, Computer Science Department, UMCP
Example: trickyReturn.cpp

• Return by value – safe but slow
• Return by pointer/reference
  – Who is supposed to call delete?
  – Be careful not to return a stale pointer/reference!
• Sometimes best to pass an extra parameter as a (non-const) reference
Arrays

Recall C-style pointer-based arrays:

```c
int x[7];  // 7 integers created
```

Recall that the variable is a pointer:

```c
x[3] is the same as *(x + 3)
```

In C++ arrays work with objects too:

```c
Cat y[7];  // 7 Cat objects instantiated using
            // default constructor
```

**Important:** It is impossible to create an array of Cats if there is no default constructor!
Dynamically Allocated Arrays

When declared as an “array variable”, the size of the array must be a constant:

Cat x[7];     // works fine
Cat x[size];  // will not compile

For dynamic allocation, use new:
Cat *x = new Cat[size];  // works fine

Be sure to give the memory back later:
delete [] x;  // Strange syntax!
new/delete Summary

• Every object allocated with `new` must eventually be de-allocated with `delete`
• Every array allocated with
  `new foo[size]` must eventually be de-allocated with `delete[]`

Important: You can’t mix them!
Example: vectorExample.cpp

Arrays are annoying. Use the `vector` template instead.

- Very similar to `ArrayList` in Java
- Don’t forget `#include<vector>`
- Use `push_back` to add an element to the end
- Use `[]` (overloaded nicely)
- To use effectively, you’ll need to learn more about templates and iterators!
Collections: Java vs. C++

In Java we collect references:

```java
ArrayList<Cat> a = new ArrayList<Cat>();
Cat c = new Cat();
a.add(c);
Cat d = a.get(0);
```

How many Cat objects were instantiated?
Collections: Java vs. C++

In C++ we collect **objects** (not references to objects):

```cpp
vector<Cat> v;
Cat c;
v.push_back(c);
Cat d = v[0];
```

How many Cat objects were instantiated?

Is there a way to avoid making copies (more like Java)? Sure:

```cpp
vector<Cat *> v;
```

(It is impossible to create a collection of references.)
Transition....

It’s time for:

• Project #1 – where you write a bunch of global functions that manipulate strings and vectors

• In lecture, we’ll begin learning about C++ object-oriented programming and writing classes!
# Class Lingo: Java vs. C++

<table>
<thead>
<tr>
<th>Java</th>
<th>C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance variable</td>
<td>Data member</td>
</tr>
<tr>
<td>Instance method</td>
<td>Member function</td>
</tr>
<tr>
<td>Static variable</td>
<td>Static variable</td>
</tr>
<tr>
<td>Static method</td>
<td>Static function</td>
</tr>
<tr>
<td>---</td>
<td>external (global) variable</td>
</tr>
<tr>
<td>---</td>
<td>(global) function</td>
</tr>
</tbody>
</table>
Ominous Foreshadowing...

Did you notice how C++ tries to treat objects and primitives exactly the same way?

This causes us great pains when writing classes!