# Languages and Data Types III

Spring 2014

Carola Wenk

# ~~Functions~~ Methods

```
public int increment(int i) {
    return i+1;
}

public void printHello() {
    System.out.println("Hello");
}

}
```

We must also declare the types of not only variables, but also of functions (called methods in Java).

# Everything is a Class…

```
public class Hello {

public static void main(String[] args){
  System.out.println(“Hello World”);
}
}
```

A key difference between Python and Java is that, while Python allows “optional” class declarations, in Java everything is a class.

That is, we cannot just execute a series of statements as in Python. Instead, all program execution occurs through the invocation of a class “instance”.

# Program Structure

```
import A, B, C

def f(x1, x2, ...):
...

def g(y1, y2, ...):
  ...


print “hello world!”

def h(z1, z2, ...):
...

print “goodbye world!”
```

```
import A, B, C;

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!”)
}
}
```

In Java, “everything is a class” so programs are initiated in the `main` method of a class, and class files are “executed.”

# Java Runtime System

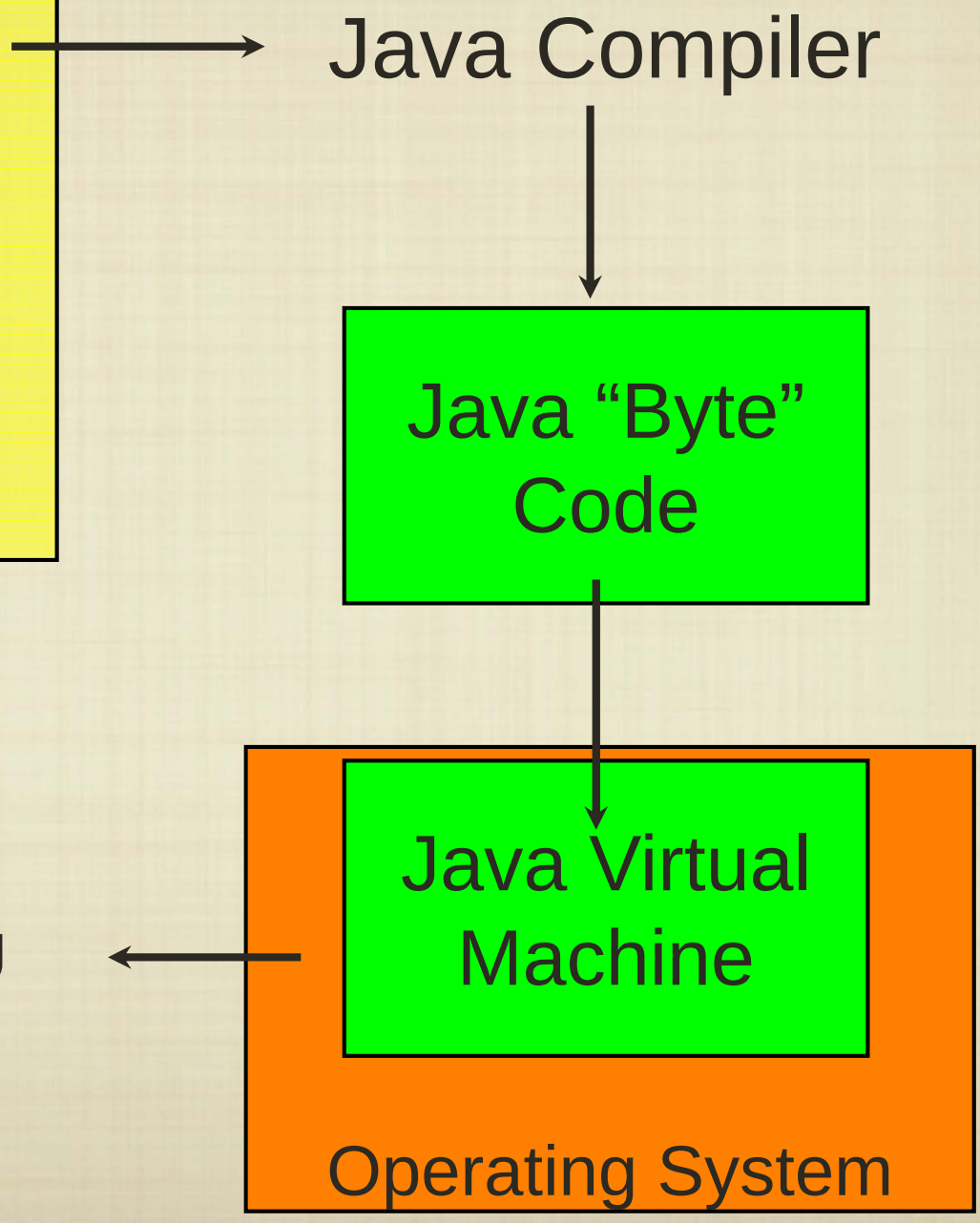
```
import --;

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!")
}
}
```



# Data Types and Classes

- Primitive types (`int`, `double`,…) provide basic functionality, but we can also declare our own types.

- An abstract data type is defined by a data structure together with functionality. “Classes” in Python and Java provide a means to define abstract data types.

```
class MyType:
<variables>

def __init__(...):
    ...

<methods>


X = MyType(...)
```

```
class MyType {
<attributes>

public MyType(...) {
...
}

<methods>

<optional main method>

}
```

```
MyType X = new MyType(...)
```

# Data Types and Classes

```
class Counter:
  # __value

  def __init__(self):
    self.__value=0

  def increment(self):
    self.__value += 1

  def getValue(self):
    return self.__value

c = Counter()
c.increment()
c.increment()
print c.getValue()
```

```
public class Counter{
  private int value;
  public Counter(){
       value=0;
  }

  public void increment(){
      value++;
  }

  public int getValue(){
    return value;
  }
}
```

```
class Tester {
  public static void main(String[]
                              args){
    Counter c = new Counter();
    c.increment();
    c.increment();
    System.out.println(c.getValue());
  }
}
```

Typically, variables of a user-defined type (or class) are called instances of that type.

# Data Types and Classes

```
public class Counter{
  private int value;
  public Counter(){
       value;
  }

  public void increment(){
      value++;
  }

  public int getValue(){
    return value;
  }
}
```

```
class Tester {
  public static void main(String[]
                              args){
    Counter c = new Counter();
    c.increment();
    c.increment();
    System.out.println(c.getValue());

    # Error, because value is private
    # c.value = 5
  }
}
```

In Java, we can restrict user access to a class by using the `public` and `private` keywords.

# Data Types and Classes

```
public class Counter{

  public Counter(){

  }

  public void increment(){

  }

  public int getValue(){

  }
}
```

```
class Tester {
  public static void main(String[]
                              args){
    Counter c = new Counter();
    c.increment();
    c.increment();
    System.out.println(c.getValue());

  }
}
```

Information hiding allows the user to assume that a class meets certain specifications without worrying about the underlying implementation.

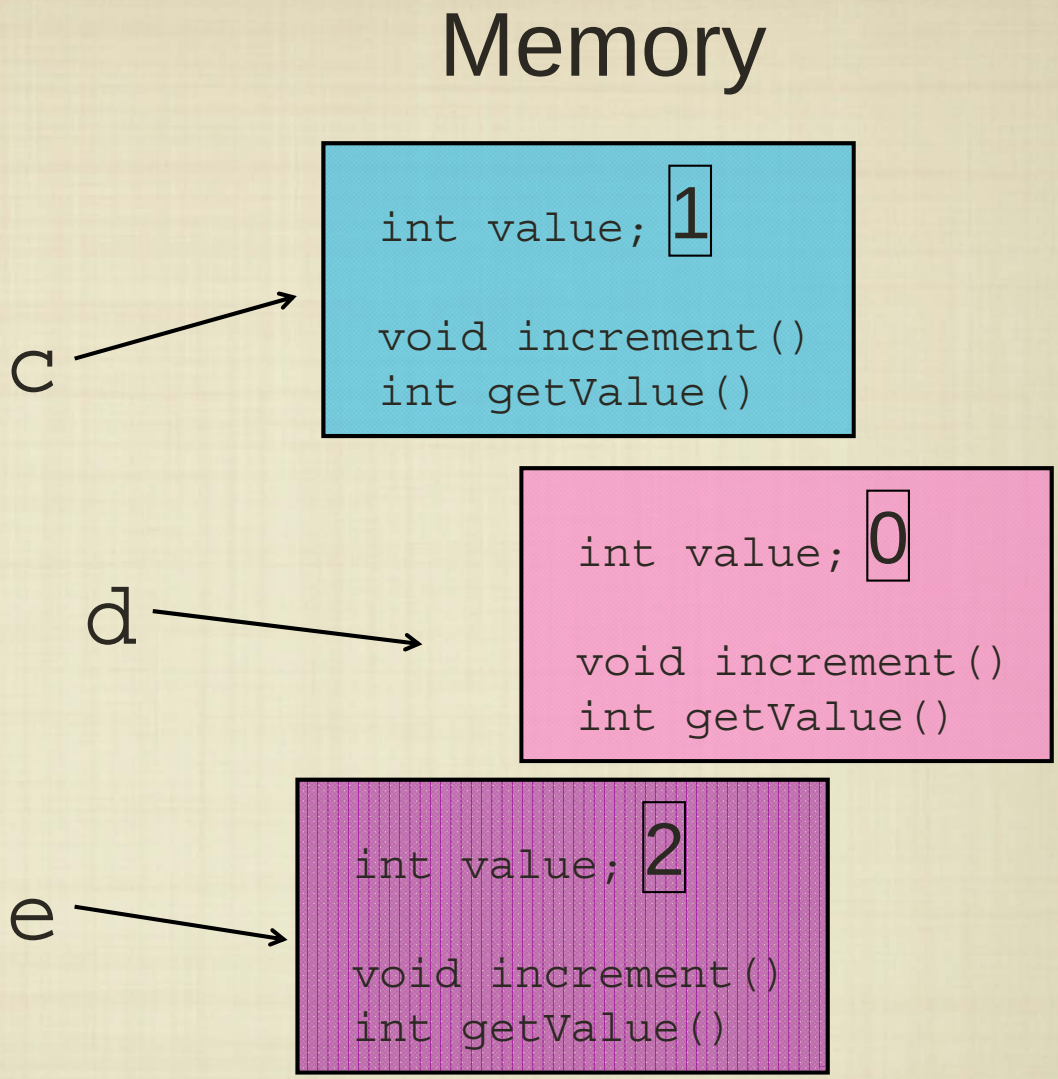
This is the principle behind so-called “APIs” and is critical in allowing us to take a top-down approach to problem solving.

# Class vs. Instances

```
Class Counter
  int value;

  void increment()
  int getValue()
```

```
Counter c = new Counter();
c.increment();
Counter d = new Counter();
Counter e = new Counter();
e.increment();
e.increment();
```

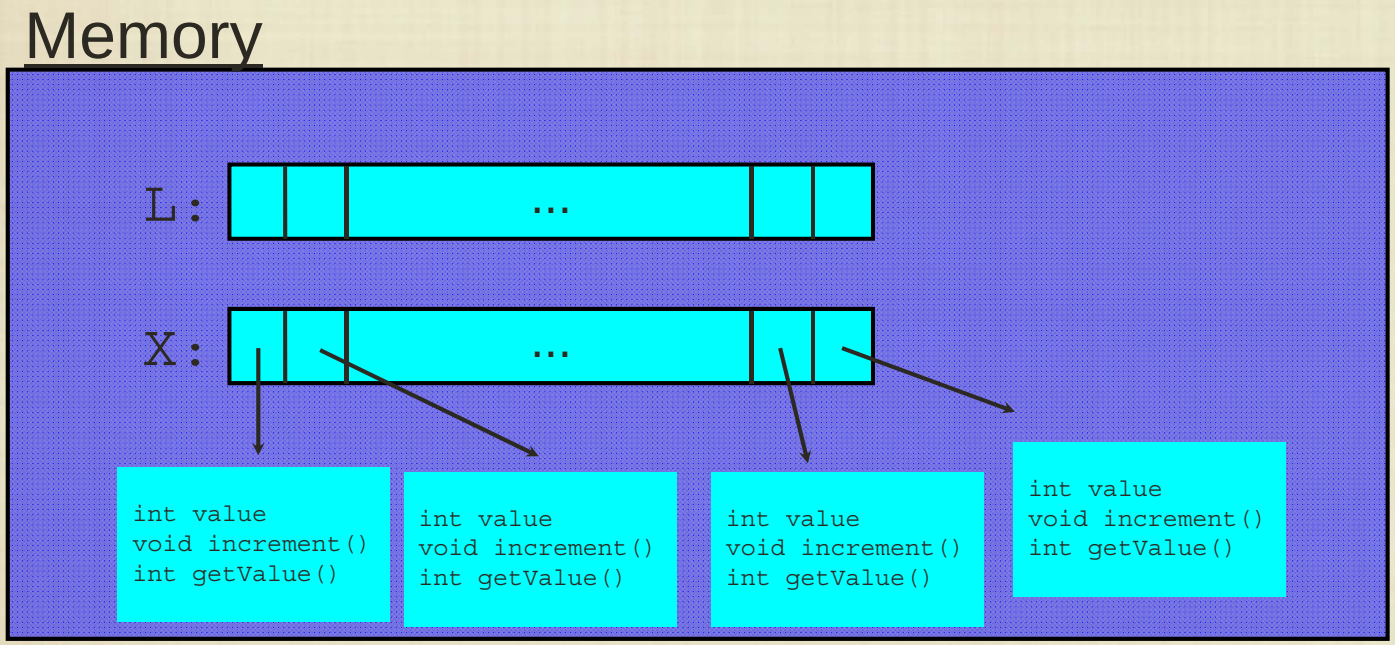



- A class is a “blueprint” that provides the definition of a type
- An instance of a class conforms to the given type definition, but has its own set of (non-static) attributes and methods, whose access is defined by how they are declared (i.e., `public` versus `private`).

# Arrays in Java

- Arrays in Java (of any type) are declared as follows:

```
int L[] = new int[25];
MyType X[] = new MyType[100];
```




User-defined types are stored by reference, while primitive types are stored by value.