CSE 1710

Lecture 2

Announcements/Housekeeping

- review new info on website
- emphasize expectations
- abbreviations used:
  - JBA = Java By Abstraction (the textbook)
  - PT = Programming Tip
  - IMD = In More Depth
- feedback re: Week #1 labs?
  - extra lab session is being offered TODAY
    - Tuesday, Sept 11, 4:30-6:00pm

Gentle reminder to these students…

Abou Daher, Serena
Du, Yong Bin
Gonzalez, Paula
Julien, Michel Junior
Leung, Matthew
Okazaki, Keegan Makoto
Tang, Si Shuang
Valle-Garay, Alejandro

The assigned reading was sec 1.1 and 1.2 (pp. 1-24)

Who completed the readings?

What are the take-aways?

do they relate to *theory*?

do they relate to *concept*?

do they relate to *praxis*?

what do these terms mean anyway?

theory – a system of ideas intended to explain something

concept – an idea, something conceived of in the mind

praxis – the putting of theory/concepts into practice/action

can you take the concepts from Ch1 and **apply** this knowledge? (e.g., *analyze* a class definition, *troubleshoot* problems, explain the difference between class files and java files, …)

The *class* is the smallest building block in Java. [KC 1.1]

Classes are organized in *package hierarchies*.

Related classes are placed in a *subpackage*.

Classes have long names (and short names).

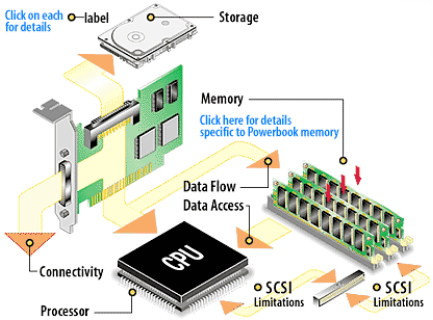
why do I care about this?

an app is made up of classes…

these classes get run by *another* app called the *virtual machine (VM)*

**the VM is not written in bytecode – it is an executable that uses machine instructions that are specific to a particular operating system/platform*
(e.g., there are different version of the VM, for each of Linux, Solaris, Mac OS X, Windows, etc)

Computer memory

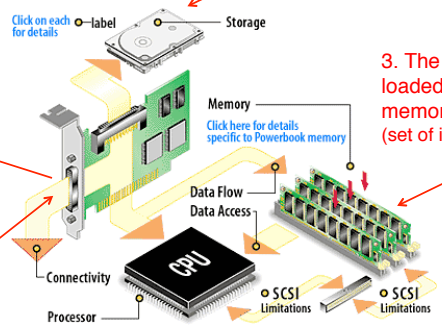


1. this is where the *.java and *.class files live (on the hard drive)

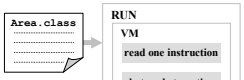
3. The VM app is loaded into memory (set of instructions)

4. app has output (to screen)

2. “run Area.class” [typed on keyboard, or mouse click on shortcut command]



[KC 1.8]

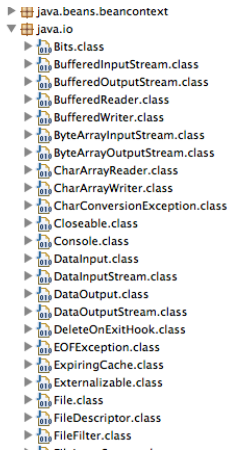


this file contains *bytecode*
We cannot (easily) read and understand it

how do we produce bytecode?

```
import java.io.PrintStream;

public class Area
{
   public static void main(String[] args)
   {
         PrintStream output;
         output = System.out;
         int width;
         width = 8;
         int height = 3;
         int area = width * height;
         output.println(area);
   }
}
```

Classes are written using a **coding style**.

[KC 1.2]

why?


```
import java.io.PrintStream;
public class Area
{public static void main(String[] args)
{PrintStream output;output = System.
          out;int width;width = 8;int height = 3;int
area = width * height;output.println(area);}}
```

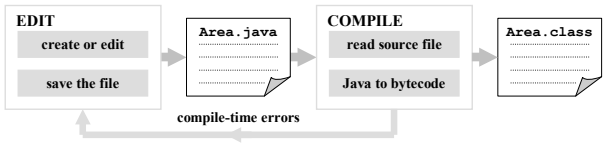
```
import java.io.PrintStream;
public class Area
{public static void main(String[] args)
{PrintStream OUTPUT; OUTPUT = System.
          out;int x;x= 8;int height = 3;int
cost = x * height; OUTPUT.println(cost);}}
```


The compiler does not care about whitespace.

[KC 1.6]

what is whitespace?

why do I care how the compiler works?




```
import java.io.PrintStream;
public class Area
{public static void main(String[] args)
{PrintStream OUTPUT; OUTPUT = system.
          out;int x;x= 8;int height = 3;int
cost = x * height; OUTPUT.println(cost);}}
```

The compiler cares about case. [KC 1.6]

why do I care how the compiler works?

Classes get defined. Their definition consists of a class header followed by a class body. [KC 1.3]

Methods get defined. Their definition consists of a method header followed by a method body.

get defined by whom? and for whom?

A block is something sandwiched between two curly braces.

how many blocks?


```
import java.io.PrintStream;

public class Area
{
   public static void main(String[] args)
   {
        PrintStream output;
        output = System.out;
        int width;
        width = 8;
        int height = 3;
        int area = width * height;
        output.println(area);
   }
}
```

```
import java.io.PrintStream;

public class Area
{
   public static void main(String[] args)
   {
        PrintStream output;
        output = System.out;
        int width;
        width = 8;
        int height = 3;
        int area = width * height;
        output.println(area);
   }
}
```



```
import java.io.PrintStream;

public class Area
{
   public static void main(String[] args)
   {
        PrintStream output;
        output = System.out;
        int width;
        width = 8;
        int height = 3;
        int area = width * height;
        output.println(area);
   }
}
```


The body of the class `Area` contains one method definition.

show this is true…


```
import java.io.PrintStream;

public class Area
{
   public static void main(String[] args)
   {
        PrintStream output;
        output = System.out;
        int width;
        width = 8;
        int height = 3;
        int area = width * height;
        output.println(area);
   }
}
```

```
import java.io.PrintStream;

public class Area
{
   public static void main(String[] args)
   {
        PrintStream output;
        output = System.out;
        int width;
        width = 8;
        int height = 3;
        int area = width * height;
        output.println(area);
   }
}
```


[KC 1.4]

There is a thing called a *statement*. Statements are delimited by semicolons (unless we are dealing with a header).

why do I need to recognize where the statements are?


```
import java.io.PrintStream;

public class Area
{
   public static void main(String[] args)
   {
        PrintStream output;
        output = System.out;
        int width;
        width = 8;
        int height = 3;
        int area = width * height;
        output.println(area);
   }
}
```


[KC 1.5]

Comments can be found in documentation or internal.

who reads comments anyways?

what is usability vs correctness?
how do comments relate to these concepts?

Usability – how easy is the app to use? how learnable is it? how steep is the learning curve?

Correctness – does the app do what it says it will do?

so how do comments relate to the concept of usability?

a useable app is *intuitive* to use – the user shouldn't have to read a pile of external documentation to use an app

a correct app does what is says it will do (and an app states what it does in its external documentation)

external documentation refers to things like the API, user manuals, FAQs, and other such documents (NOT the comments within the code itself)

[KC 1.7]

What are all of the lexical elements?

do I need to recognize these?


```
import java.io.PrintStream;

public class Area
{
   public static void main(String[] args)
   {
         PrintStream output;
         output = System.out;
         int width;
         width = 8;
         int height = 3;
         int area = width * height;
         output.println(area);
   }
}
```

Keywords, Identifiers, Literals, Operators, Separators

Suppose I have 4 bits.

How many unique representations do I get with these 4 bits?

Task #1: come up with a scheme to represent the age of a car (in years)

Task #2: come up with a scheme to represent hourly pay rates

So the very same 16 representations can be used for two different schemes: ages in years or dollar rates.

So too can the same 4 bytes be used for two different schemes: a big set of integers or a big set of real numbers

the sets are the same size, but the representations are mapped out differently

look at the representation scheme for `short` and for `char`

both use 2 bytes (or 65536 unique representations)

- `short` represents integers from -32768 to 32767
- `char` represents a code in the Unicode table, ranging from 0 to 65535

the sets are the same size, but the representations are mapped out differently

consider the arrangement of 0's and 1's that represent the *integer number* `4`

&

consider the arrangement of 0's and 1's that represent the *real number* `4.0`

are the 0's and 1's the same in both cases?

[KC 1.6]

If your app wants to store a value, it needs to say so in a statement…

version 1: “Let’s store a value”

```
my $fee = 25;
my $payrateperhour = 31.55;
```

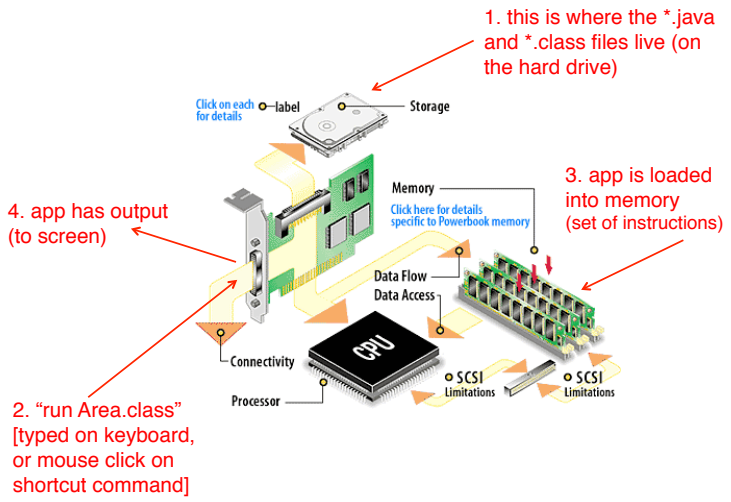
(this is how Perl works)

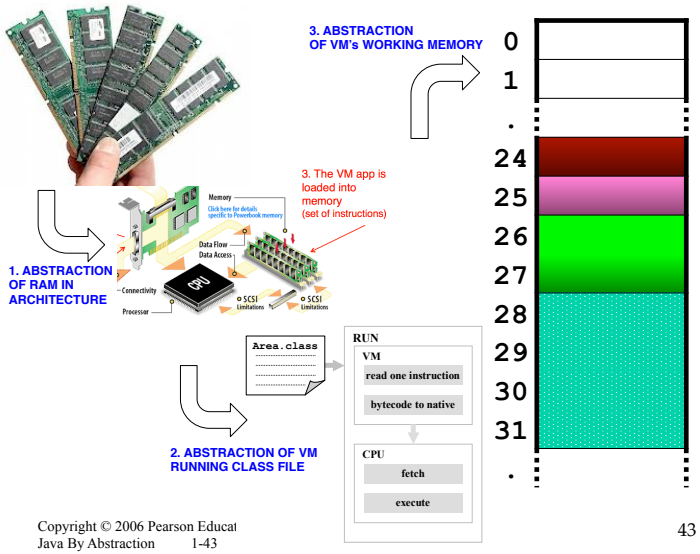
version 2: “Let’s store *this particular type* of value”

```
int fee = 25
double payRatePerHour = 31.55;
```

(this is how Java works)

which of these versions is **strongly typed**?





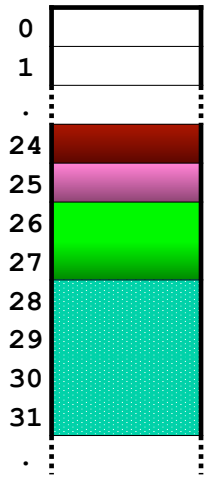
The diagram is a schematic of the VM’s working memory

1-byte block at address 24

1-byte block at address 25

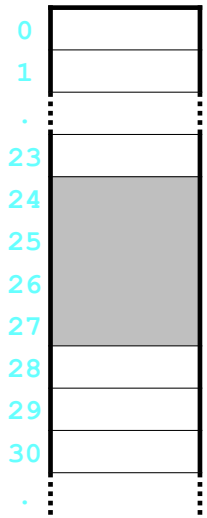
2-byte block at address 26

4-byte block at address 28



What happens when the VM sees bytecode that corresponds to this:

```
int width;
```

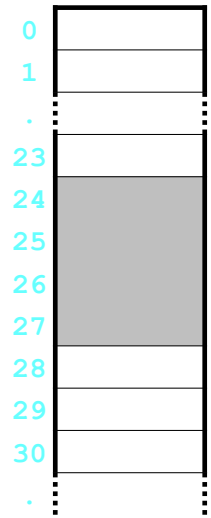


1. A block big enough to hold an `int` is allocated, e.g. a 4B block at 24
2. Its address is associated with the variable name, e.g. 24 with `width`

Note that no initialization is involved; only an association of a name with an address.

What happens when the VM sees bytecode that corresponds to this:

```
int width;
```

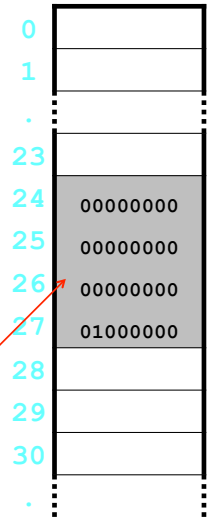


1. A block big enough to hold an `int` is allocated, e.g. a 4B block at 24
2. Its address is associated with the variable name, e.g. 24 with `width`the
3. An entry in the symbol table is made:

Identifier	Type	Block Address
width	int	24

What happens when the VM sees bytecode that corresponds to this:

```
int width = 64;
```

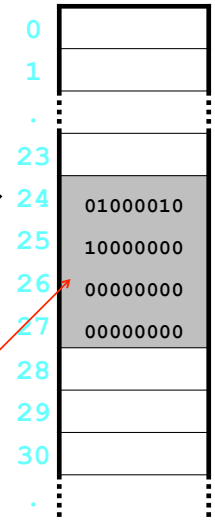


Identifier	Type	Block Address
width	int	24

representation in two's complement (scheme for integers)
(the actual representation)

What happens when the VM sees bytecode that corresponds to this:

```
float width = 64f;
```



Identifier	Type	Block Address
width	float	24

representation in IEEE-754 (scheme for floating point numbers)
(the actual representation)