Recap

- program structure

```
public class Progname {
  public static void main ( String[] args ) {
    statements
  }
}
```

- types – the *type* determines how values can be manipulated
  - numeric: `int`, `double`
  - text: `char`, `String`
  - true/false: `boolean`
- literals
  - `int` and `double`: `5`, `5.0`
  - `char` and `String`: `'5'`, `"5"`
  - `boolean`: `true`, `false`

what is the type of "`true`"?

Variables

a *variable* is a named box that can hold a value

- a box can only hold one value at a time, but that value can change
- use the box's name to refer to the value the box holds

compare –

- subtract 32 from 50, then multiply by 5/9 to get 10
- subtract 32 from the fahrenheit temperature, then multiply by 5/9 to get the celsius temperature
- subtract 32 from f, then multiply by 5/9 to get c

Statements – Variable Declaration

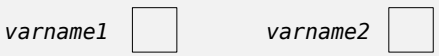
- purpose: creating and naming a box that can hold a value

`type varname;`

- semantics: create a variable named *varname* which can hold a value of type *type*

`type varname1, varname2, …;`

- semantics: create variables named *varname1*, *varname2*, etc, each of which can hold a value of type *type*



- convention is to start variable names with lowercase letters
- use camel case (first letter of each word capitalized) or _ (underscore) for multiple words

More on Variable Naming and Comments

- choose descriptive names for variables
  - the name should reflect what the variable is for
  - names should distinguish similar variables from each other
  - single-letter names are rarely descriptive
  - simply appending 1, 2, … is rarely descriptive
- include a comment with any clarifying information not apparent from the variable name

```
String name;
String name;            // the player's name

double price;
double salePrice;

double temperature;     // in degrees F
```

the comment can be omitted if there's only one name in the context of the program

Statements – Variable Declaration

Is there any difference between

```
int a, b;
```

and

```
int a;
int b;
```

?

- there is no difference in semantics
- the first form is convenient if the variables are related to each other and can be described with a single comment

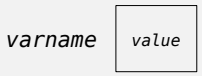
```
int width, height;    // dimensions of the field
int area;             // area of the field
```


Statements – Assignment Statement

- purpose: putting a value into a variable

varname `=` *value*`;`

- *value* can be a literal, a variable name, or an expression
- *varname* must have been declared previously
- *value* must be of a type compatible with the declared type of *varname* or else a syntax error will result
- semantics: put *value* into the variable *varname*, replacing any value that might have already been in *varname*



Statements – Output

- purpose: to display values on the screen

`System.out.print(`*value*`);`
- semantics: display *value*

`System.out.println(`*value*`);`
- semantics: display *value* following by a newline

- *value* can be a literal, a variable name, or an expression
  - use the name of a variable to refer to the value stored in that variable

these questions are asking about what is on the first, second, etc line of the output that is produced when the program is run, not what output the first, second, etc line of the program produces

```
public class Warmup1 {
  public static void main ( String[] args ) {

    System.out.println("Goodbye!");
    System.out.print("abc");

    double x;
    x = 4;

    System.out.println("x");
    System.out.println(x);

    System.out.println("Hello!");
  }
}
```

```
int a, b;
a = 5;
b = a;
a = 10;
System.out.println(b);
```

statements are instructions carried out at that moment, not statements of fact that persist into the future

it is better to read assignment statements as "a gets the value 5" rather than "a equals 5"

- what does \n mean? e.g. “hello\ngoodbye”
  - \n is the *newline character*

The Big Picture

New things introduced –

- arithmetic operators: +, -, *, /, %
- relational operators: == , !=, <, >, <=, >=
- expressions and type conversion
  - an *expression* is anything which represents or computes a value
    - a literal (e.g. 3, 8.7, “hello”, 'z')
    - a variable name (e.g. `x`, `balance`)
    - a function call (e.g. `Math.random()`)
    - one or more expressions combined by an operator (e.g. `3*x`, `“hello”+x`)
- built-in subroutines – `Math`, `String`
  - a *subroutine* is a named bundle of instructions that you can use to perform a task
    - *built-in* means “part of the standard Java libraries” so you don't have to write it yourself or do anything special in order to use it


```
public class Warmup2 {
  public static void main ( String[] args ) {

    double x;
    x = 4;

    System.out.println("x");
    System.out.println(x);
    System.out.println(x*10+1);
    System.out.println(Math.pow(4,2));

    String s;
    s = "Greetings!";

    System.out.println(s.length());
    System.out.println("Hello!"+s);
  }
}
```

```
int a, b;
a = 14;
b = 4;
System.out.println(a/b);
```


Arithmetic Operators

- arithmetic operators: +, -, *, /, %
  - `/` works differently for `int` and `double`
    - if either *a* or *b* are doubles, `a/b` is what you would expect
    - if both *a* and *b* are integers, `a/b` does *integer division* and is the number of whole times *b* divides *a*
  - `%` is mod (modulus) – `a % b` is the remainder when *a* is divided by *b*
    - typically only applied to `int`, but it does work for `double` as well

`8.5/2.5` is 3.4
`2.5/12.5` is 0.2
`7.5/2.5` is 3.0
`8.0/10.0` is 0.4
`8.0/10` is 0.4
`8/10.0` is 0.4

`8/2` is 4
`15/2` is 7
`8/10` is 0

`7 % 5` is 2
`23 % 5` is 3
`5 % 13` is 5

- string concatenation: +

`“hi”+” “+”there”` is “hi there”
`“gr”+8` is “gr8”