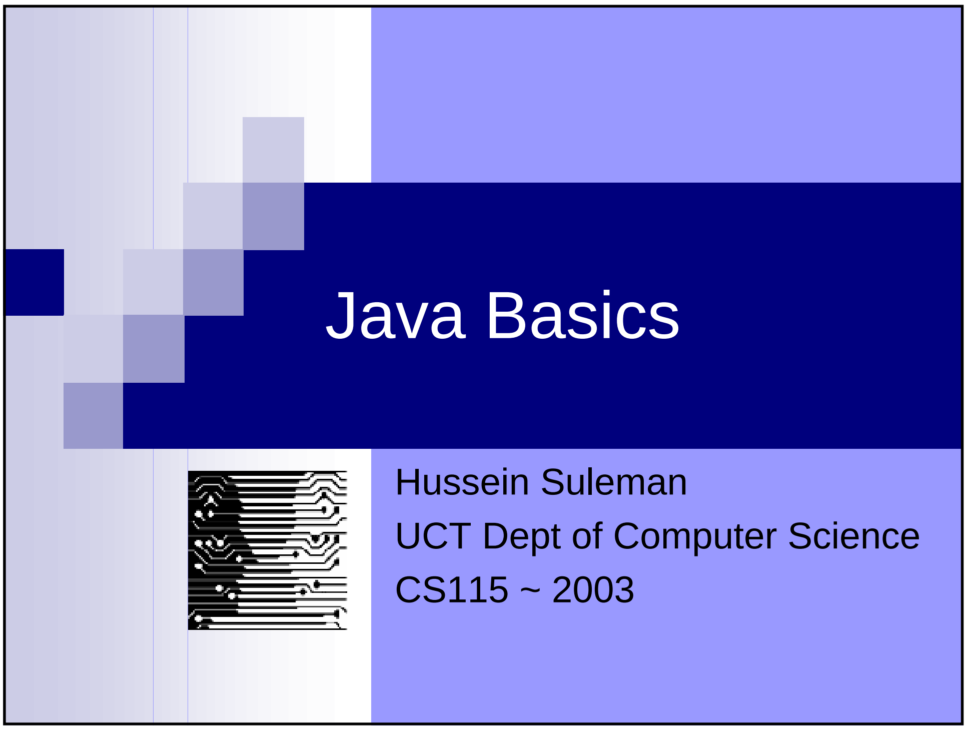
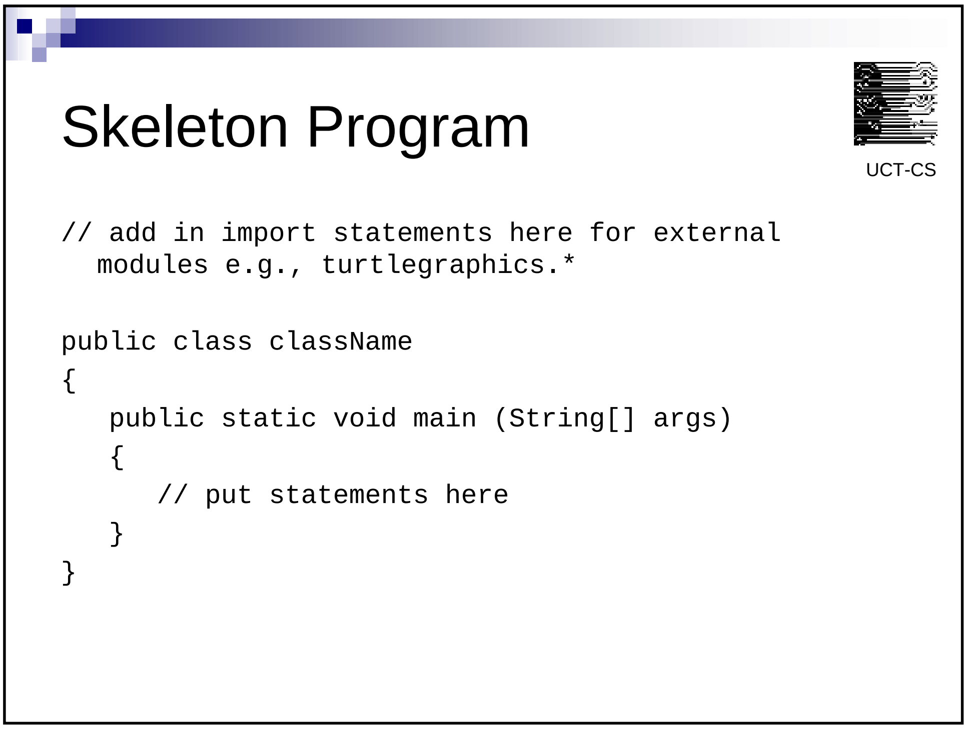


Java Basics

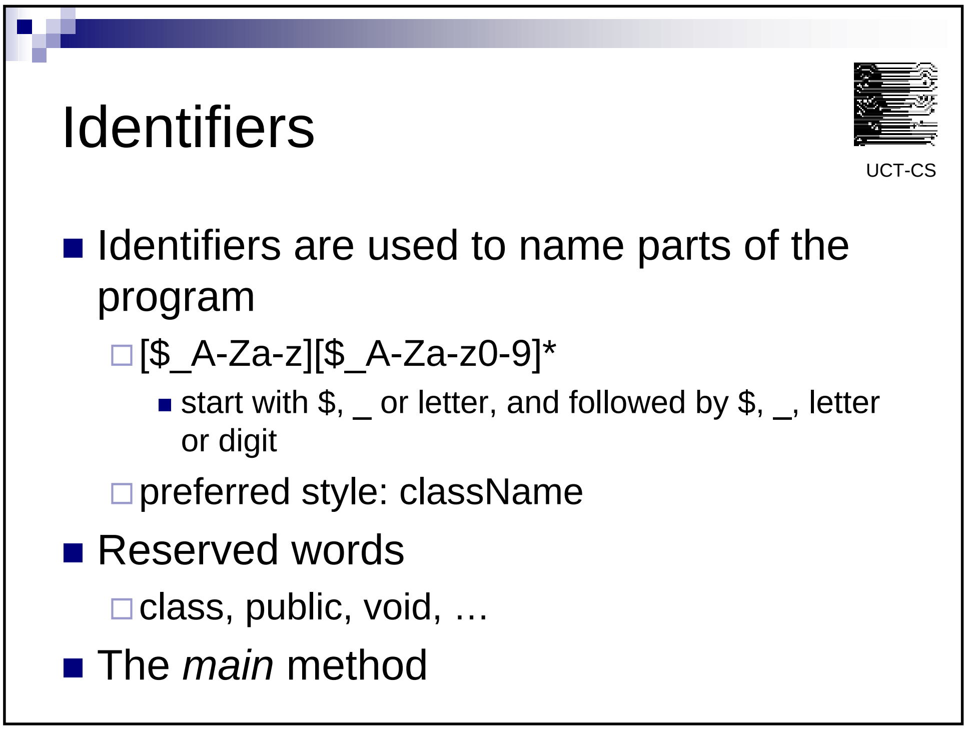
Hussein Suleman
UCT Dept of Computer Science
CS115 ~ 2003



UCT-CS

Skeleton Program

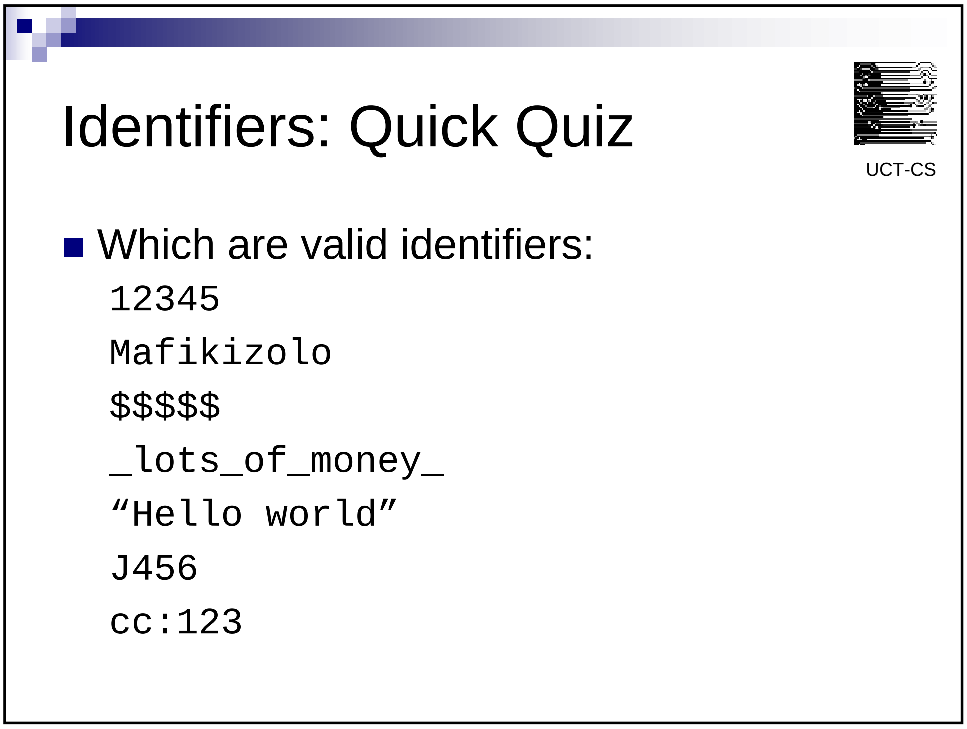
```
// add in import statements here for external
  modules e.g., turtlegraphics.*

public class className
{
   public static void main (String[] args)
   {
      // put statements here
   }
}
```



Identifiers

UCT-CS

- Identifiers are used to name parts of the program
  - [$_A-Za-z][$_A-Za-z0-9]*
    - start with $, _ or letter, and followed by $, _, letter or digit
  - preferred style: className
- Reserved words
  - class, public, void, …
- The *main* method



Identifiers: Quick Quiz

UCT-CS

- Which are valid identifiers:

```
12345
Mafikizolo
$$$$$
_lots_of_money_
“Hello world”
J456
cc:123
```

Classes and Methods

UCT-CS

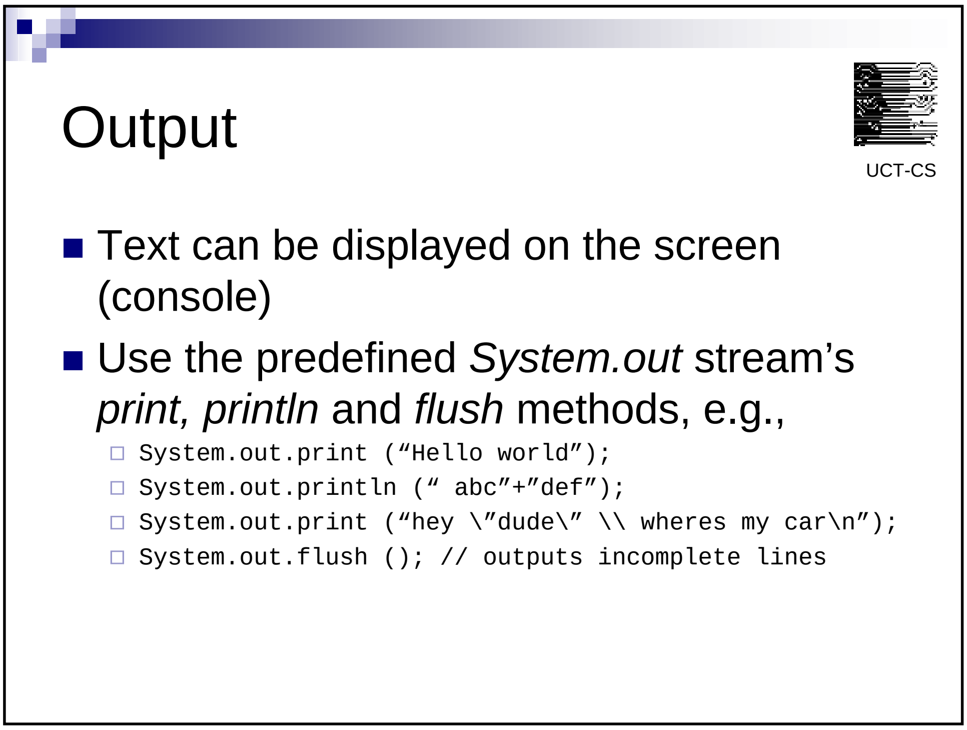
- Class defines the template for creating objects
- Methods are sets of statements defined within a class
  - □ e.g., main
- To use a class, create an object of that type
  - □ e.g., Turtle t = new Turtle ();
- To use a method, call it from its object with “dot” notation
  - □ e.g., t.move (400);

Syntax

UCT-CS

- Semicolons after every statement
- Case-sensitivity
  - □ STUFF vs stuff vs STuff vs stUFF
- Everything after // is a comment

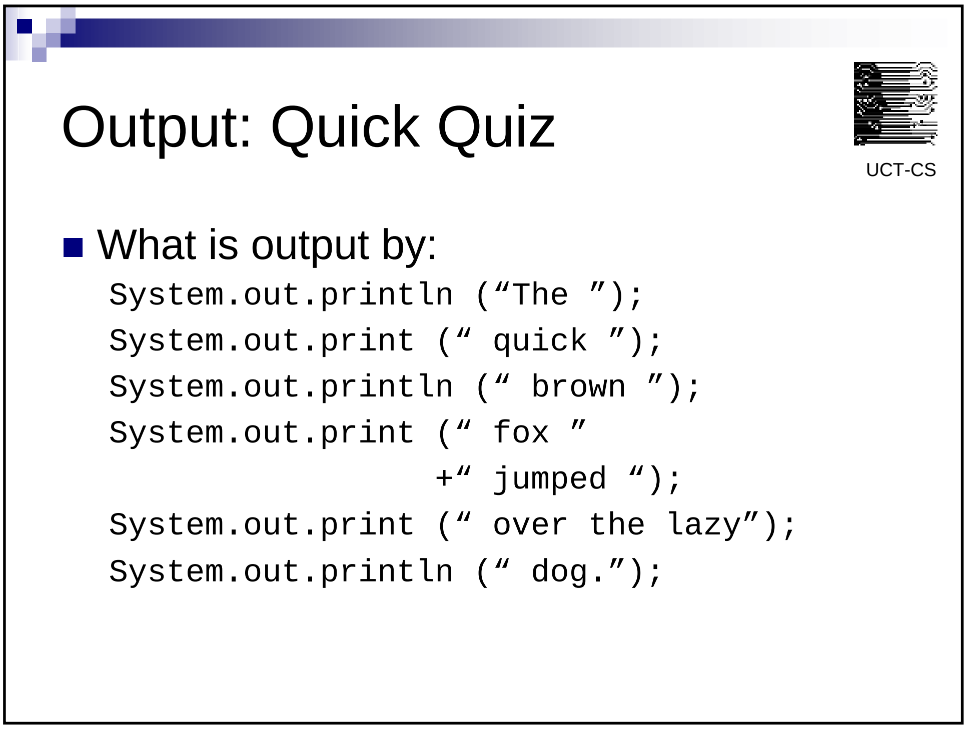
```
// a sample method
public void test
{
   Turtle t = new Turtle (); // create turtle
}
```



Output

UCT-CS

- Text can be displayed on the screen (console)
- Use the predefined *System.out* stream's *print, println* and *flush* methods, e.g.,
  - □ `System.out.print (“Hello world”);`
  - □ `System.out.println (“ abc”+”def”);`
  - □ `System.out.print (“hey \”dude\” \\ wheres my car\n”);`
  - □ `System.out.flush (); // outputs incomplete lines`

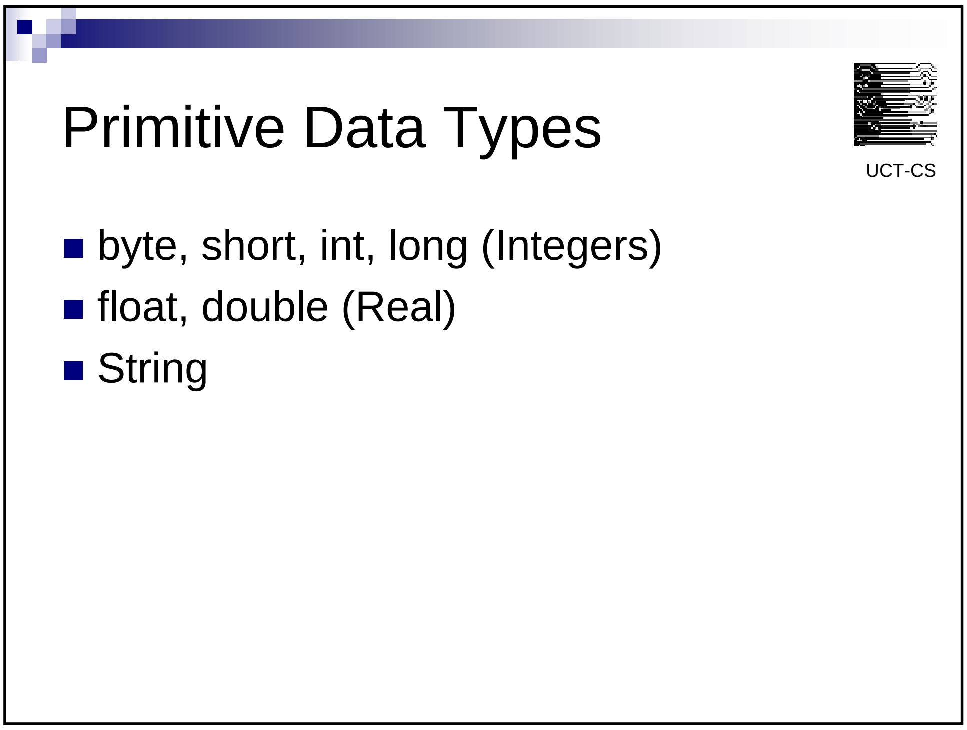


Output: Quick Quiz

UCT-CS

- What is output by:

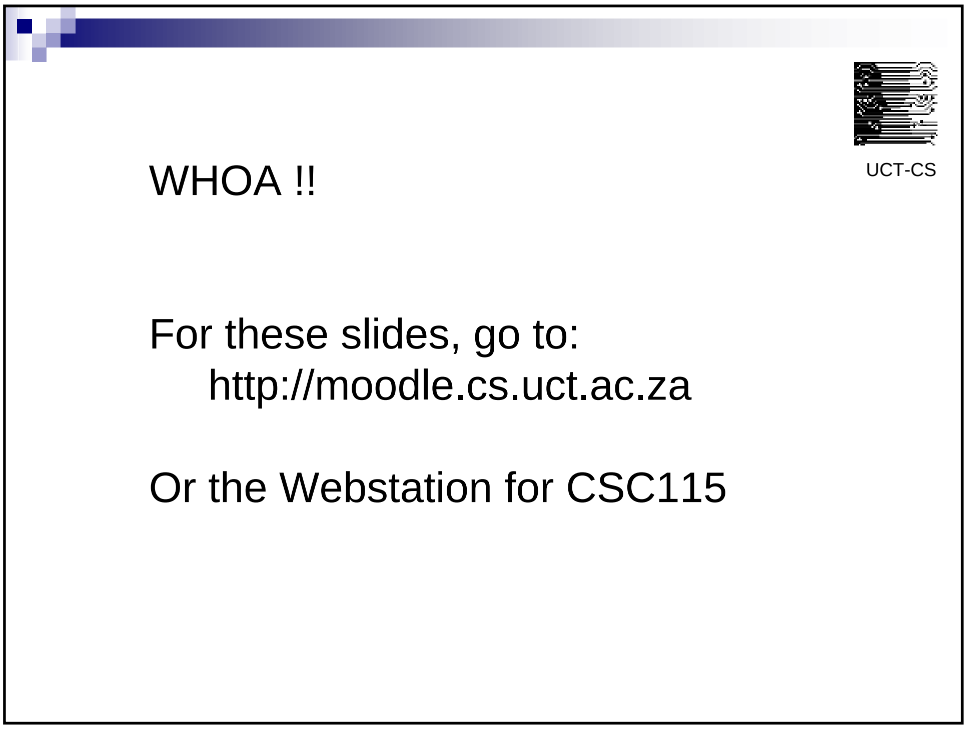
```
System.out.println (“The ”);
System.out.print (“ quick ”);
System.out.println (“ brown ”);
System.out.print (“ fox ”
                 +“ jumped “);
System.out.print (“ over the lazy”);
System.out.println (“ dog.”);
```



UCT-CS

Primitive Data Types

- byte, short, int, long (Integers)
- float, double (Real)
- String



UCT-CS

WHOA !!

For these slides, go to:
http://moodle.cs.uct.ac.za

Or the Webstation for CSC115

Integers: Literals

UCT-CS

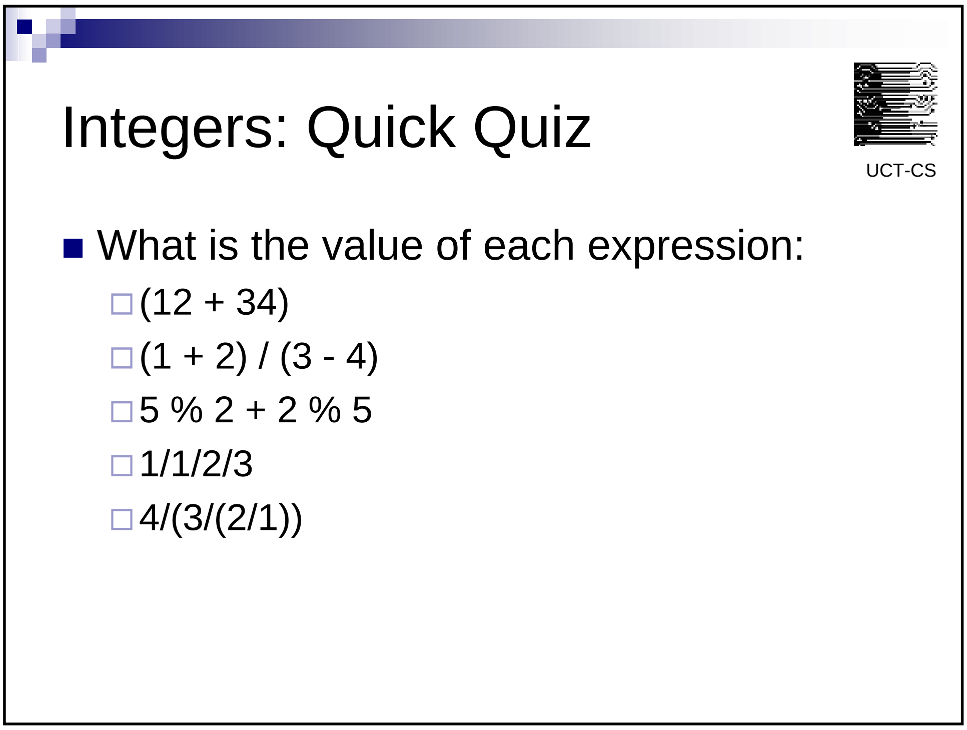
- Integer literals are converted to strings if at least one literal is a string
  - □ `System.out.print (“No:” + 12);`
    - No: 12
  - □ `System.out.print (12 + 13);`
    - 25
  - □ `System.out.print (“No:” + (12 + 13));`
    - No: 25

Integers: Expressions

UCT-CS

- Common operations

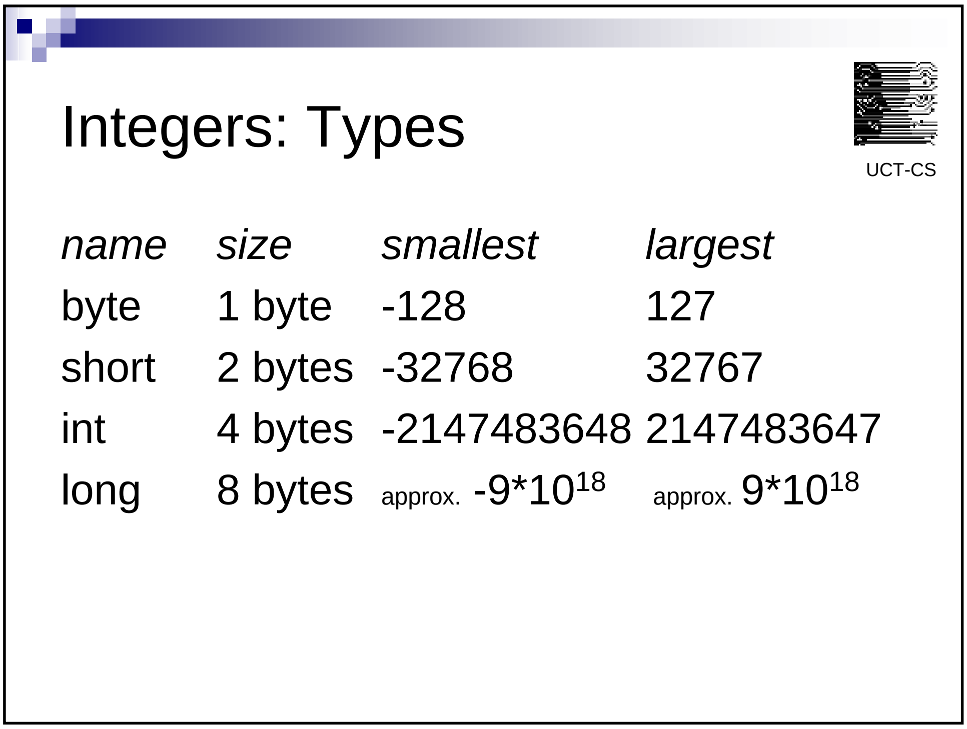
  + (plus), - (minus), / (divide), * (times), % (mod)
- 11 + 11 / 2 = 16 … how ?
  - □ precedence of operators
    - high: ( )
    - middle: * / %
    - low: + -
  - □ left associative if equal precedence
  - □ integer operations when both “operands” are integers



UCT-CS

Integers: Quick Quiz

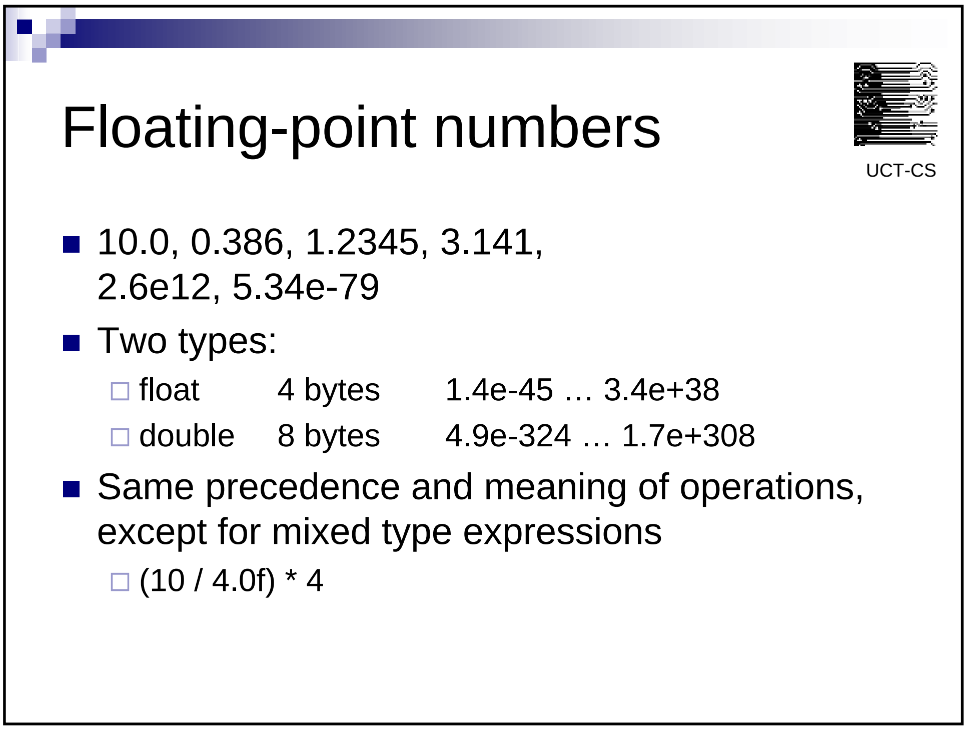
- What is the value of each expression:
  - □ (12 + 34)
  - □ (1 + 2) / (3 - 4)
  - □ 5 % 2 + 2 % 5
  - □ 1/1/2/3
  - □ 4/(3/(2/1))



UCT-CS

Integers: Types

name	*size*	*smallest*	*largest*
byte	1 byte	-128	127
short	2 bytes	-32768	32767
int	4 bytes	-2147483648	2147483647
long	8 bytes	approx. $-9*10^{18}$	approx. $9*10^{18}$



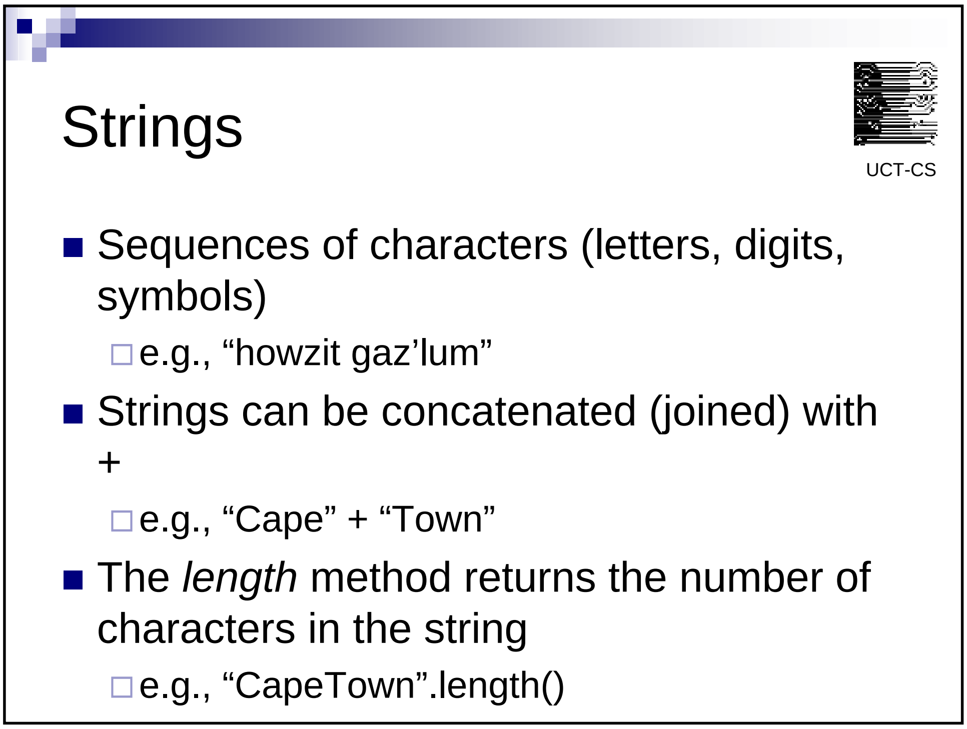
UCT-CS

Floating-point numbers

- 10.0, 0.386, 1.2345, 3.141, 2.6e12, 5.34e-79
- Two types:

□ float	4 bytes	1.4e-45 … 3.4e+38
□ double	8 bytes	4.9e-324 … 1.7e+308

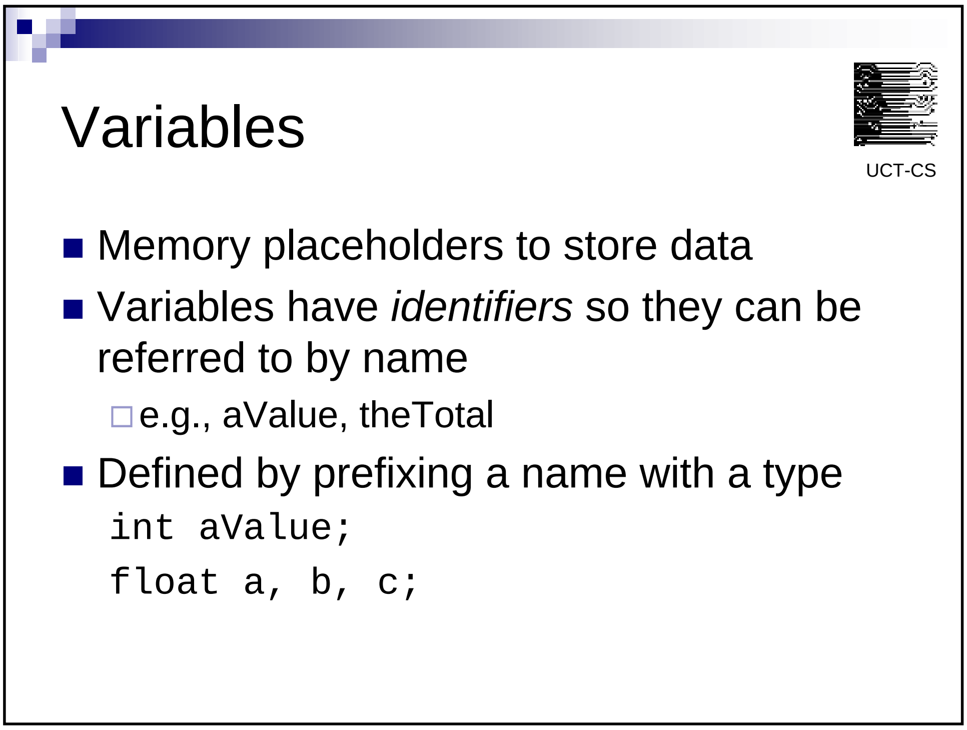
- Same precedence and meaning of operations, except for mixed type expressions
  - □ (10 / 4.0f) * 4



UCT-CS

Strings

- Sequences of characters (letters, digits, symbols)
  - □ e.g., “howzit gaz’lum”
- Strings can be concatenated (joined) with +
  - □ e.g., “Cape” + “Town”
- The *length* method returns the number of characters in the string
  - □ e.g., “CapeTown”.length()

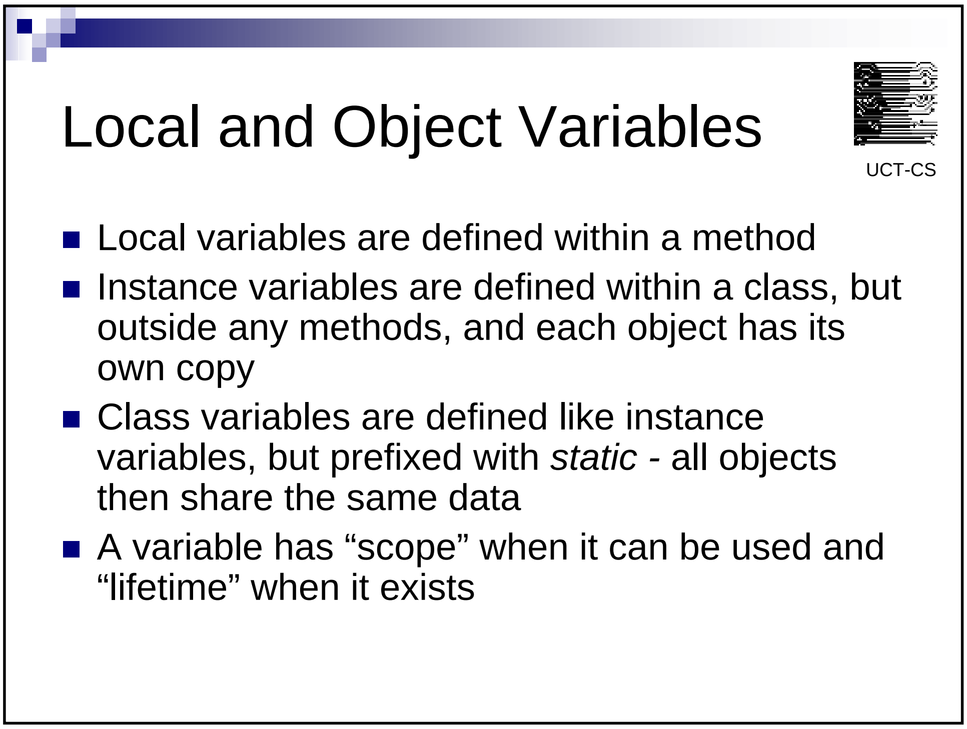


Variables

UCT-CS

- Memory placeholders to store data
- Variables have *identifiers* so they can be referred to by name
  - □ e.g., aValue, theTotal
- Defined by prefixing a name with a type

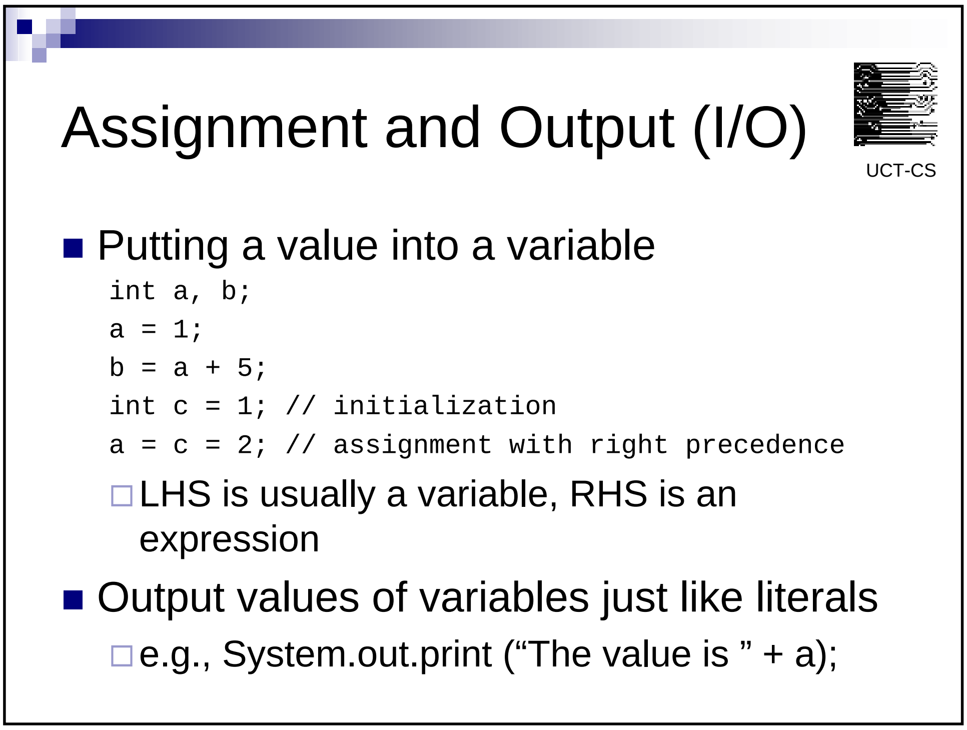
```
int aValue;
float a, b, c;
```



Local and Object Variables

UCT-CS

- Local variables are defined within a method
- Instance variables are defined within a class, but outside any methods, and each object has its own copy
- Class variables are defined like instance variables, but prefixed with *static* - all objects then share the same data
- A variable has “scope” when it can be used and “lifetime” when it exists



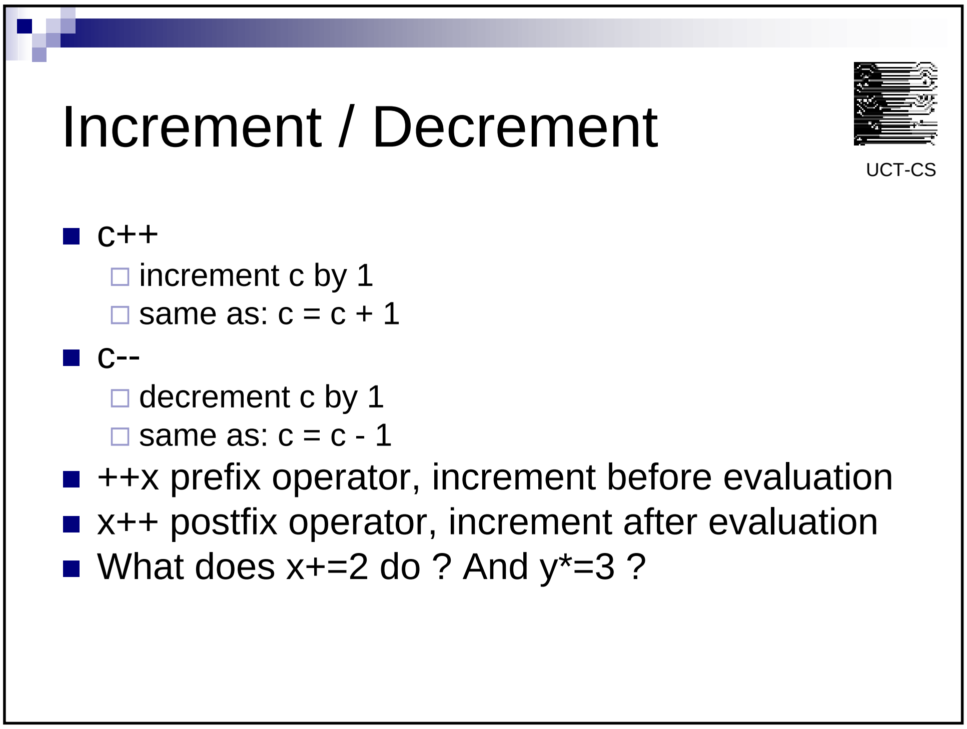
Assignment and Output (I/O)

UCT-CS

- Putting a value into a variable

```
int a, b;
a = 1;
b = a + 5;
int c = 1; // initialization
a = c = 2; // assignment with right precedence
```

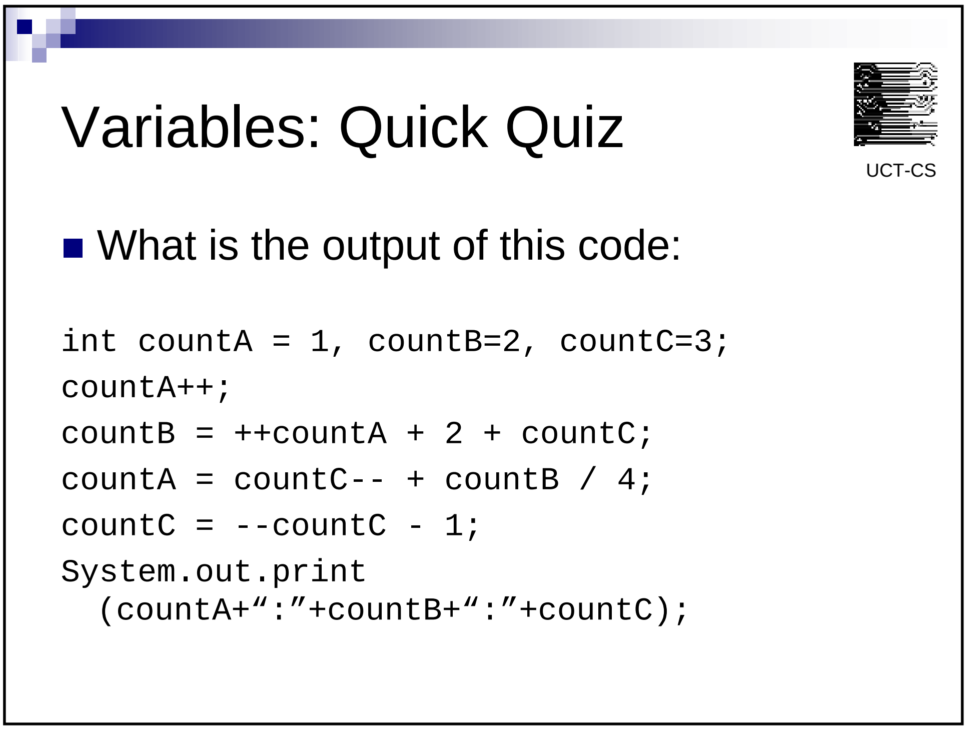
  - □ LHS is usually a variable, RHS is an expression
- Output values of variables just like literals
  - □ e.g., System.out.print (“The value is ” + a);



Increment / Decrement

UCT-CS

- c++
  - □ increment c by 1
  - □ same as: c = c + 1
- c--
  - □ decrement c by 1
  - □ same as: c = c - 1
- ++x prefix operator, increment before evaluation
- x++ postfix operator, increment after evaluation
- What does x+=2 do ? And y*=3 ?

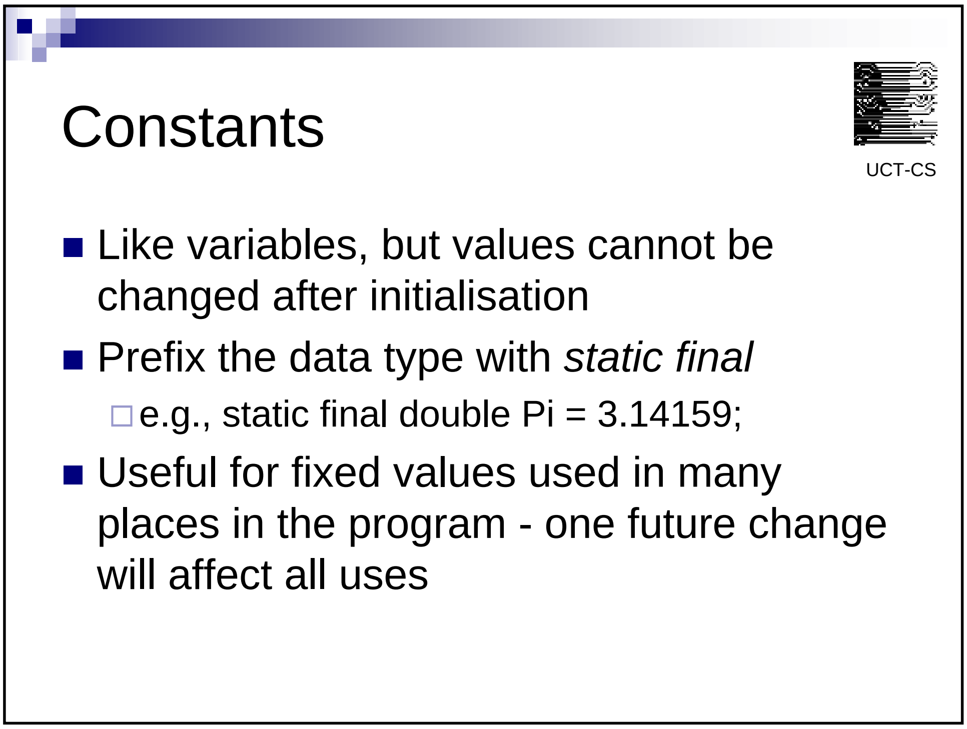


Variables: Quick Quiz

UCT-CS

- What is the output of this code:

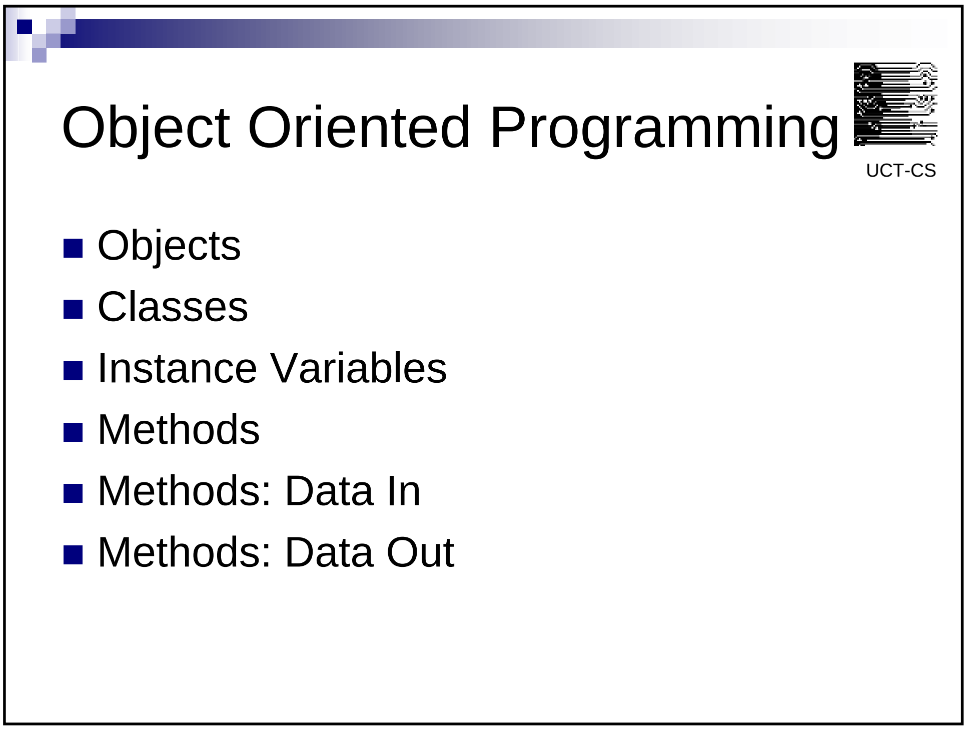
```
int countA = 1, countB=2, countC=3;
countA++;
countB = ++countA + 2 + countC;
countA = countC-- + countB / 4;
countC = --countC - 1;
System.out.print
  (countA+“:”+countB+“:”+countC);
```



Constants

UCT-CS

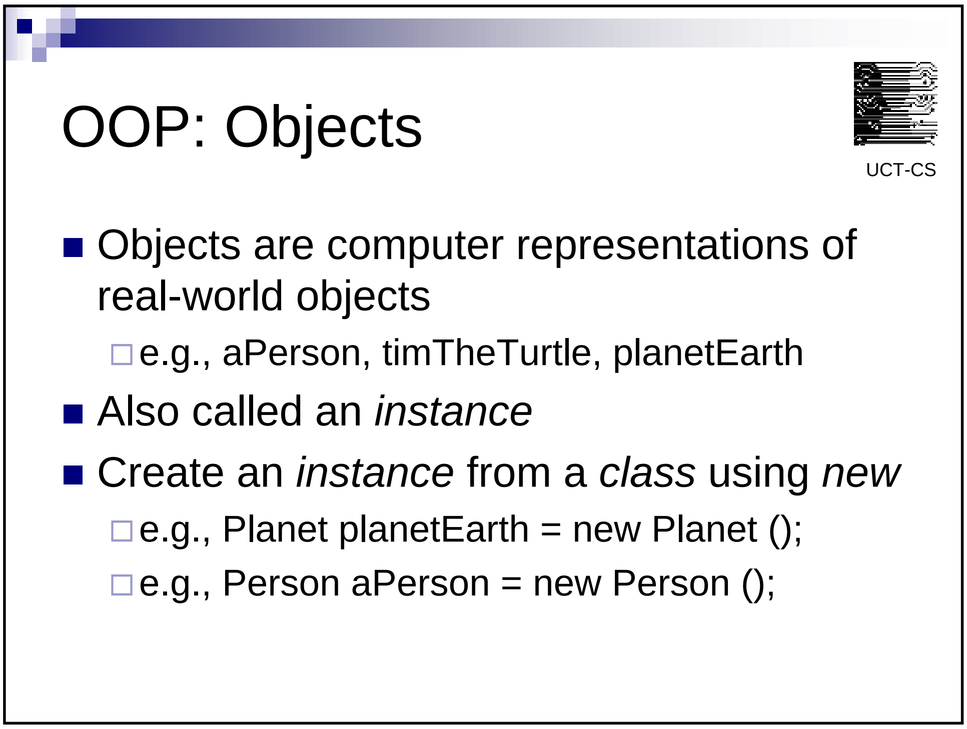
- Like variables, but values cannot be changed after initialisation
- Prefix the data type with *static final*
  - □ e.g., static final double Pi = 3.14159;
- Useful for fixed values used in many places in the program - one future change will affect all uses



Object Oriented Programming

UCT-CS

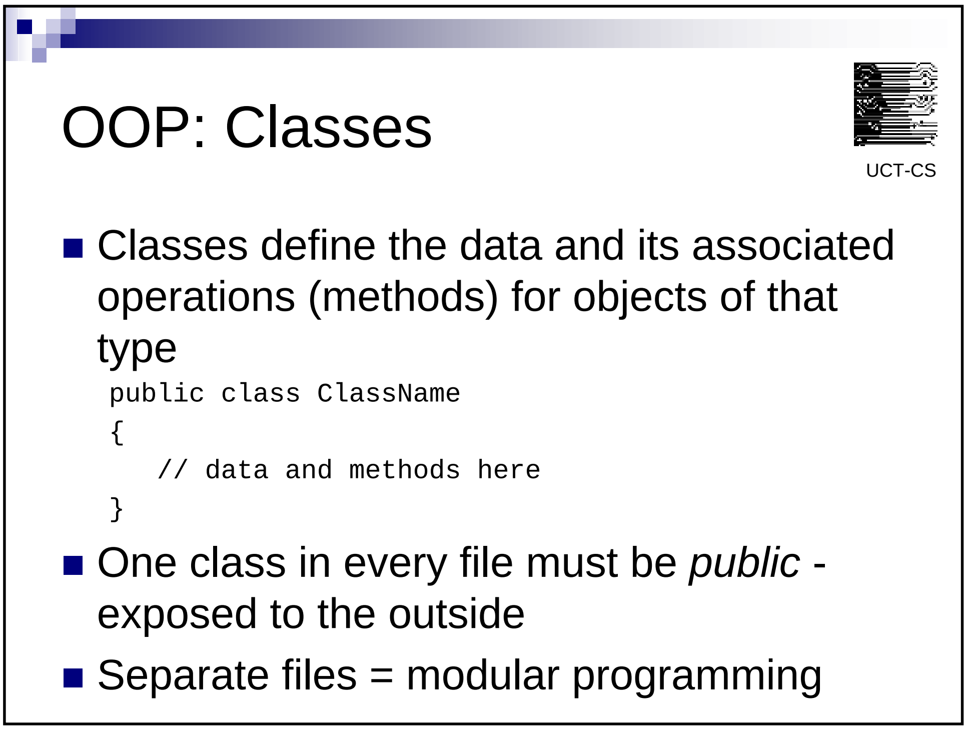
- Objects
- Classes
- Instance Variables
- Methods
- Methods: Data In
- Methods: Data Out



OOP: Objects

UCT-CS

- Objects are computer representations of real-world objects
  - e.g., aPerson, timTheTurtle, planetEarth
- Also called an *instance*
- Create an *instance* from a *class* using *new*
  - e.g., Planet planetEarth = new Planet ();
  - e.g., Person aPerson = new Person ();



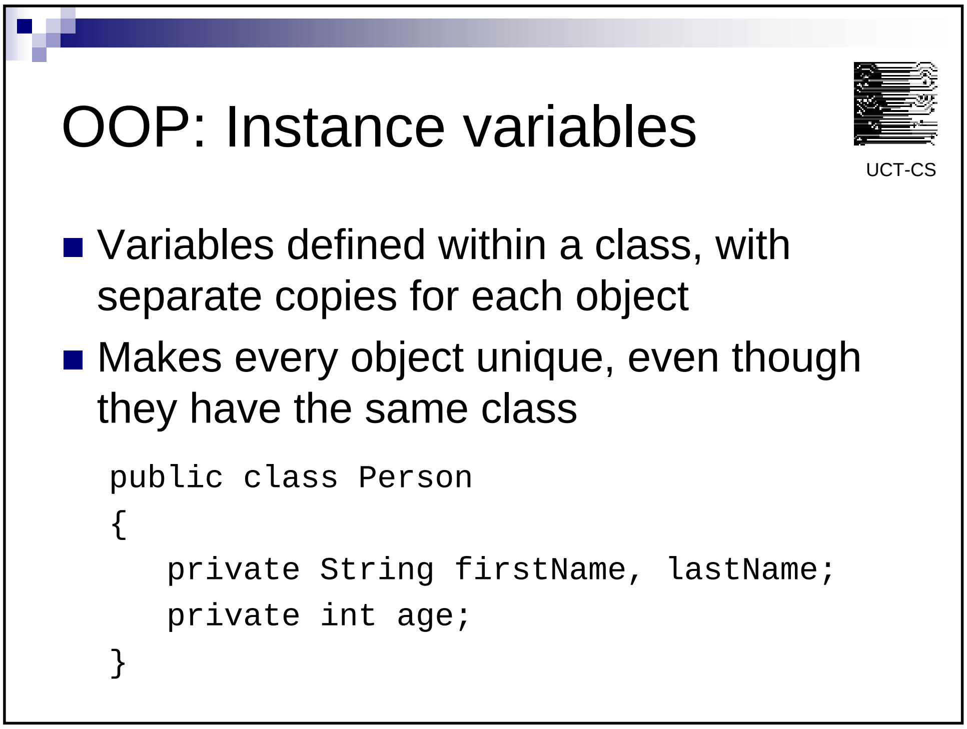
OOP: Classes

UCT-CS

- Classes define the data and its associated operations (methods) for objects of that type

```
public class ClassName
{
    // data and methods here
}
```

- One class in every file must be *public* - exposed to the outside
- Separate files = modular programming

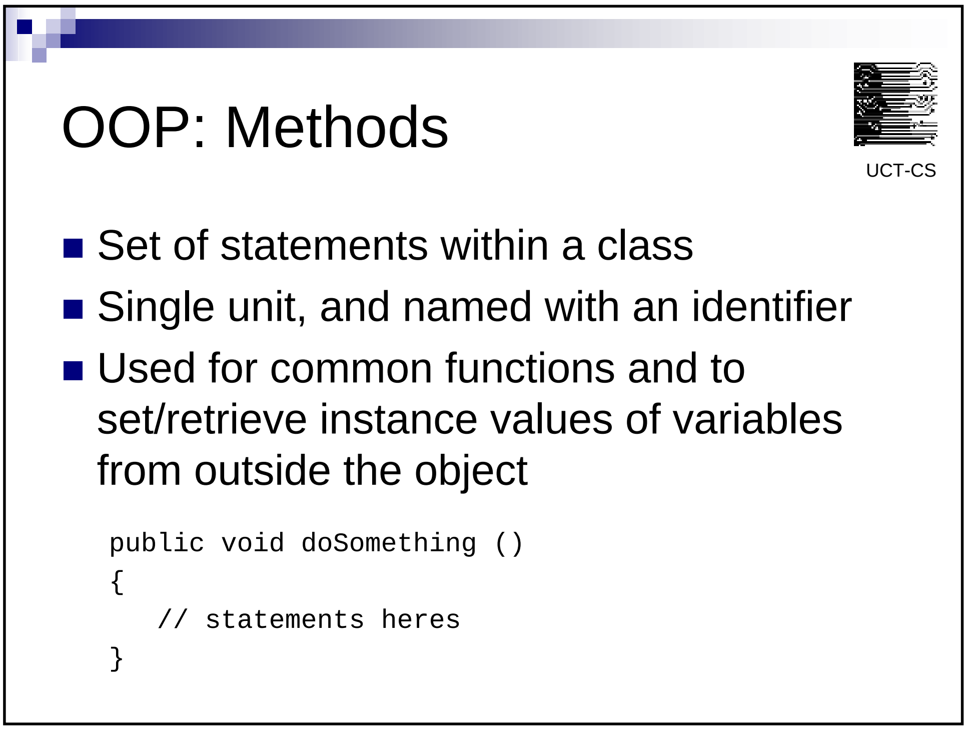


OOP: Instance variables

UCT-CS

- Variables defined within a class, with separate copies for each object
- Makes every object unique, even though they have the same class

```
public class Person
{
   private String firstName, lastName;
   private int age;
}
```

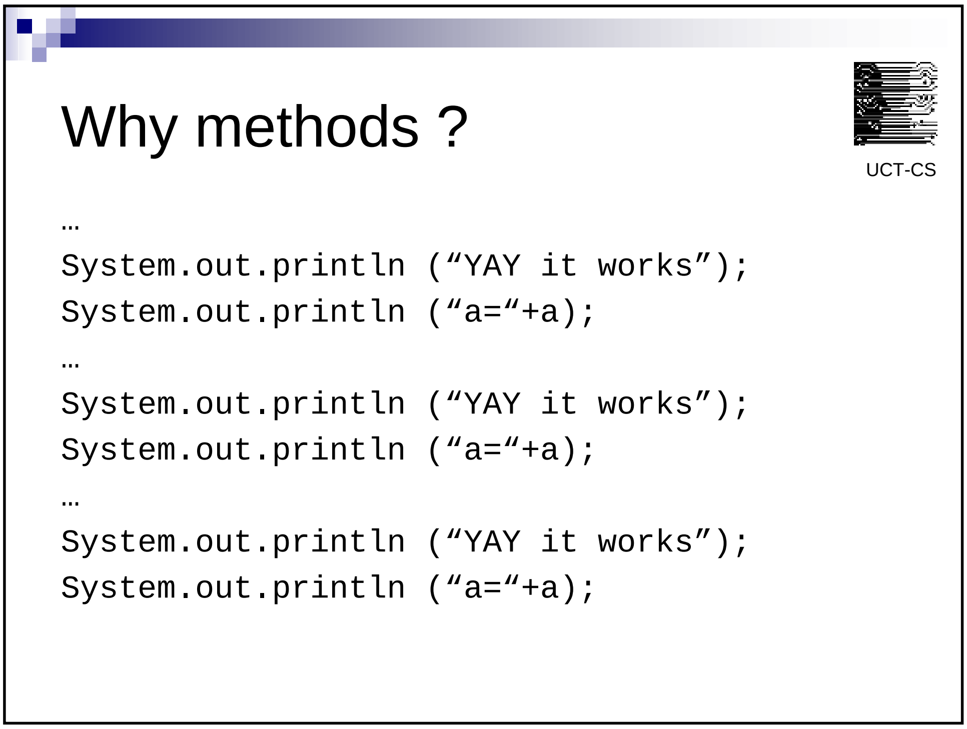


OOP: Methods

UCT-CS

- Set of statements within a class
- Single unit, and named with an identifier
- Used for common functions and to set/retrieve instance values of variables from outside the object

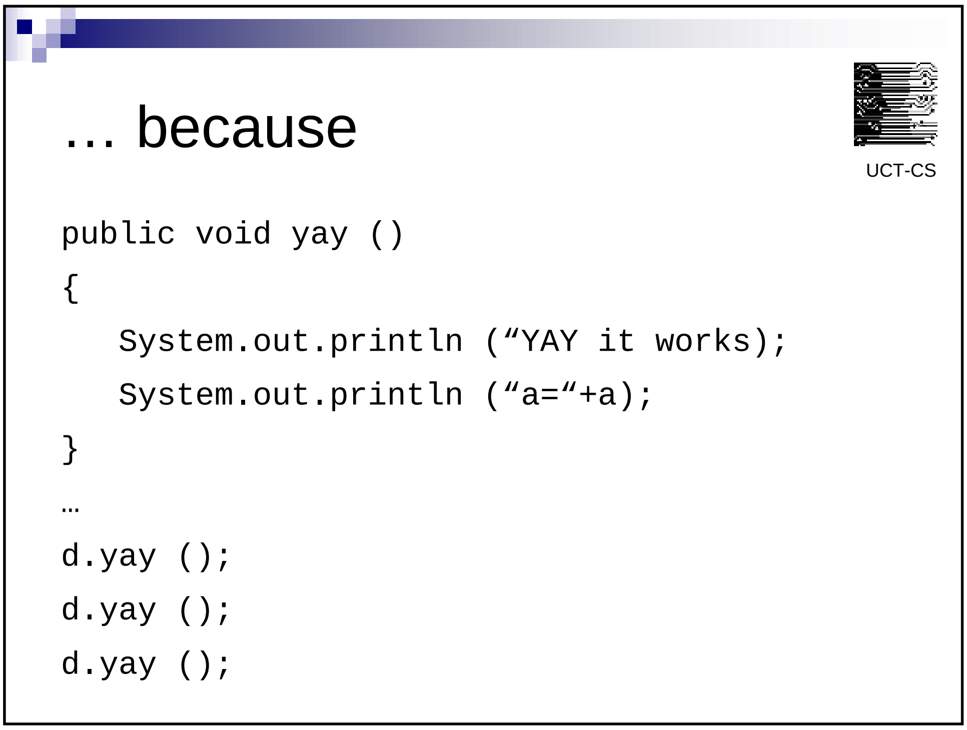
```
public void doSomething ()
{
   // statements heres
}
```



Why methods ?

UCT-CS

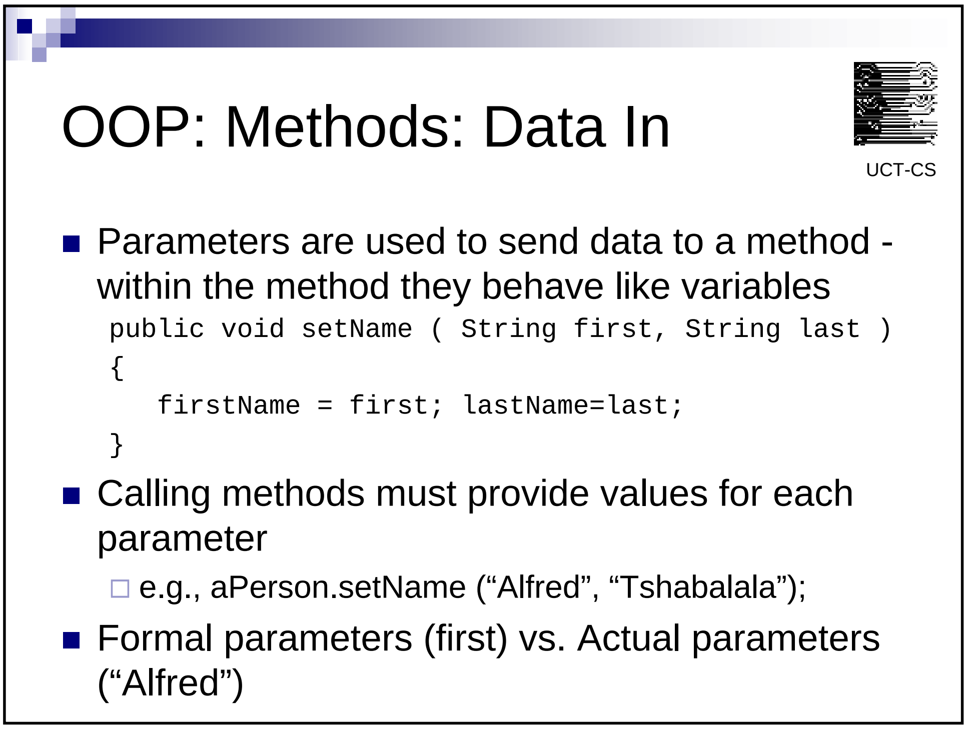
```
…
System.out.println (“YAY it works”);
System.out.println (“a=“+a);
…
System.out.println (“YAY it works”);
System.out.println (“a=“+a);
…
System.out.println (“YAY it works”);
System.out.println (“a=“+a);
```



UCT-CS

… because

```
public void yay ()
{
   System.out.println (“YAY it works);
   System.out.println (“a=“+a);
}
…
d.yay ();
d.yay ();
d.yay ();
```



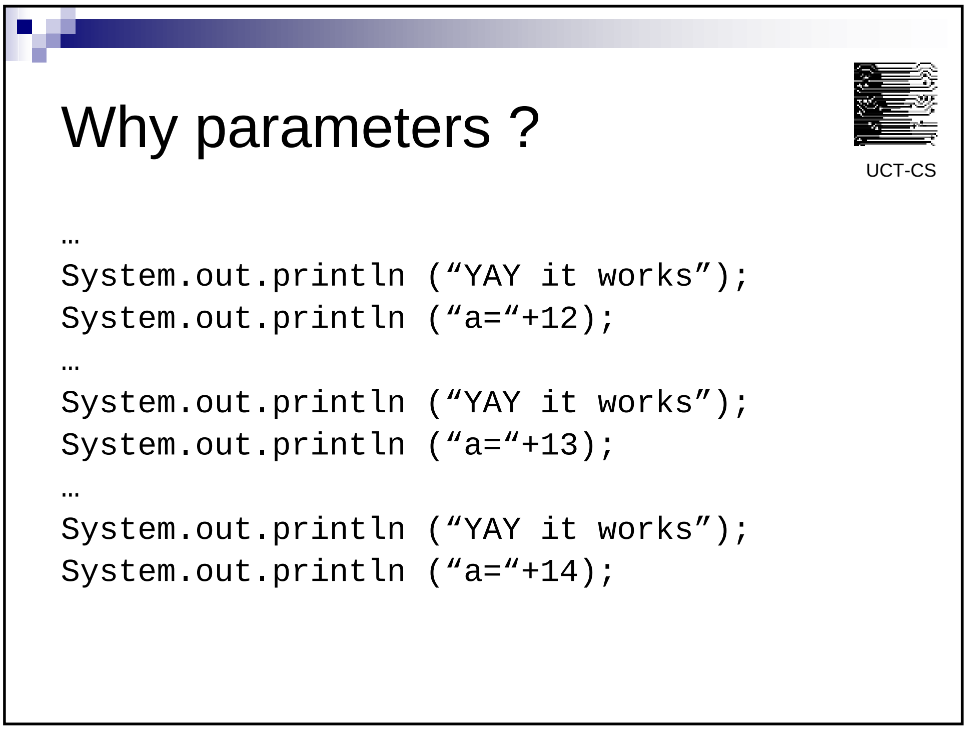
UCT-CS

OOP: Methods: Data In

- Parameters are used to send data to a method - within the method they behave like variables

```
public void setName ( String first, String last )
{
   firstName = first; lastName=last;
}
```

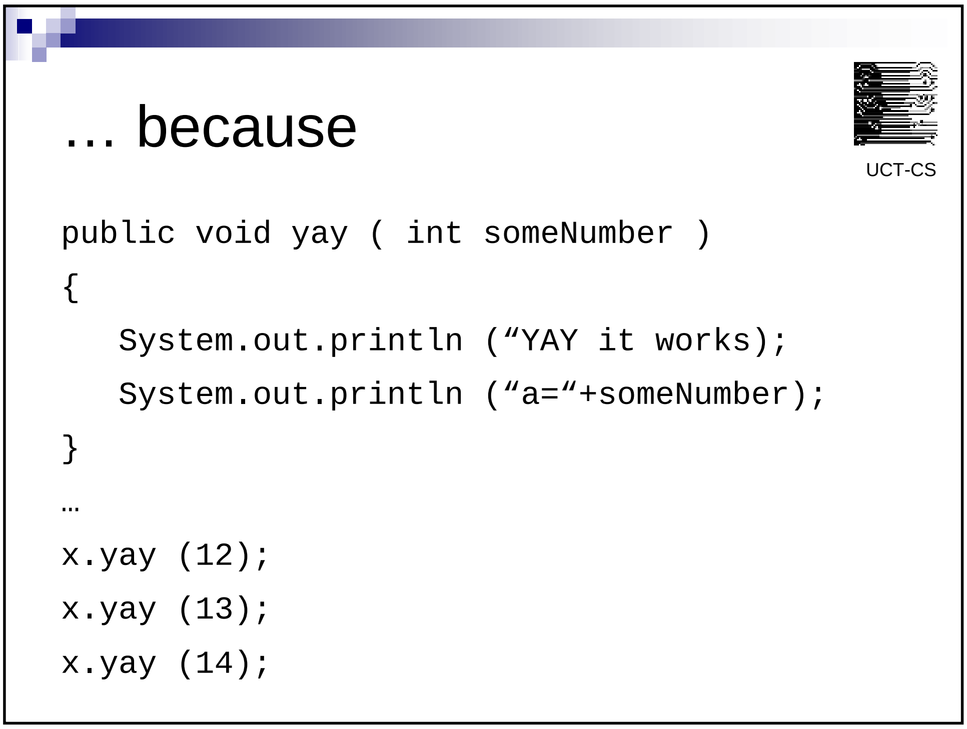
- Calling methods must provide values for each parameter
  - □ e.g., aPerson.setName (“Alfred”, “Tshabalala”);
- Formal parameters (first) vs. Actual parameters (“Alfred”)



Why parameters ?

UCT-CS

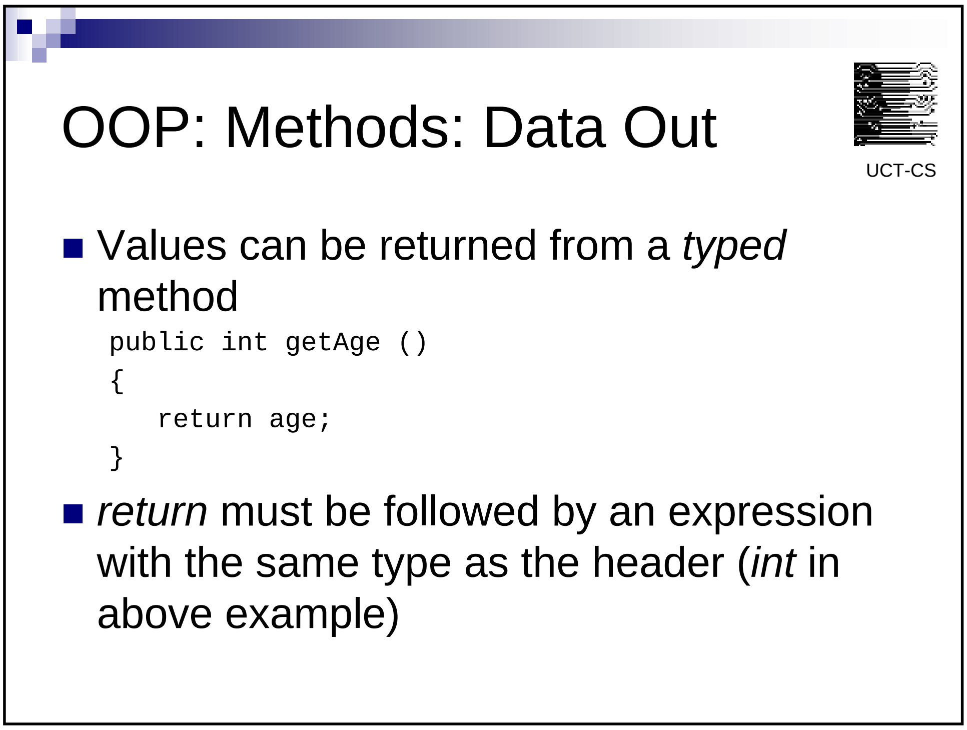
```
…
System.out.println (“YAY it works”);
System.out.println (“a=“+12);
…
System.out.println (“YAY it works”);
System.out.println (“a=“+13);
…
System.out.println (“YAY it works”);
System.out.println (“a=“+14);
```



… because

UCT-CS

```
public void yay ( int someNumber )
{
   System.out.println (“YAY it works);
   System.out.println (“a=“+someNumber);
}
…
x.yay (12);
x.yay (13);
x.yay (14);
```



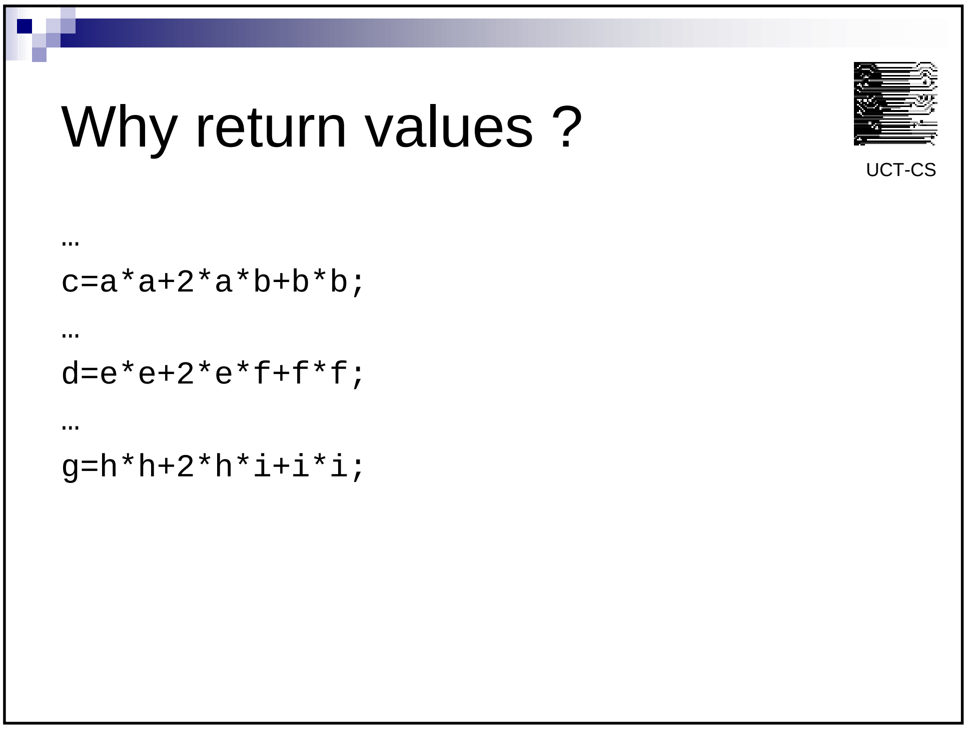
OOP: Methods: Data Out

UCT-CS

- Values can be returned from a *typed* method

```
public int getAge ()
{
   return age;
}
```

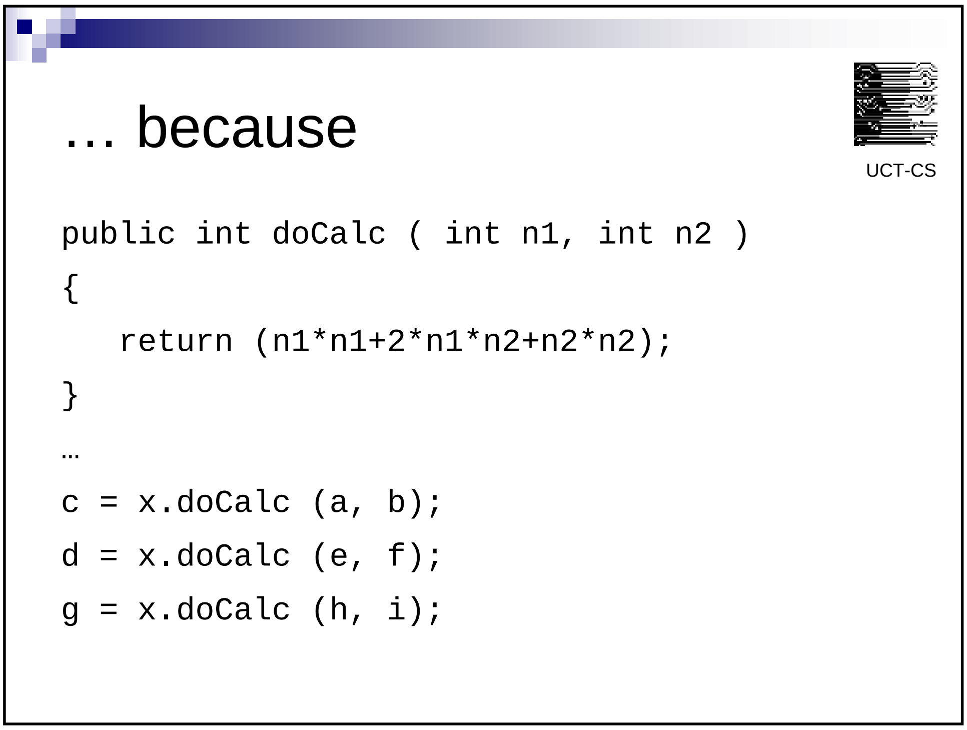
- *return* must be followed by an expression with the same type as the header (*int* in above example)



Why return values ?

UCT-CS

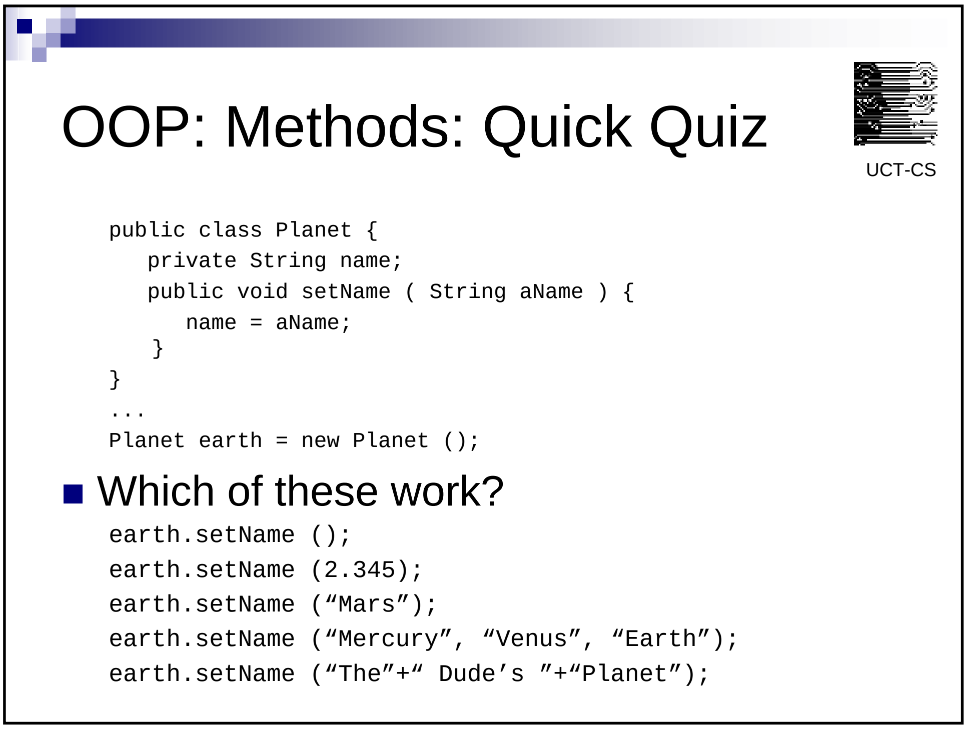
```
…
c=a*a+2*a*b+b*b;
…
d=e*e+2*e*f+f*f;
…
g=h*h+2*h*i+i*i;
```



… because

UCT-CS

```
public int doCalc ( int n1, int n2 )
{
   return (n1*n1+2*n1*n2+n2*n2);
}
…
c = x.doCalc (a, b);
d = x.doCalc (e, f);
g = x.doCalc (h, i);
```



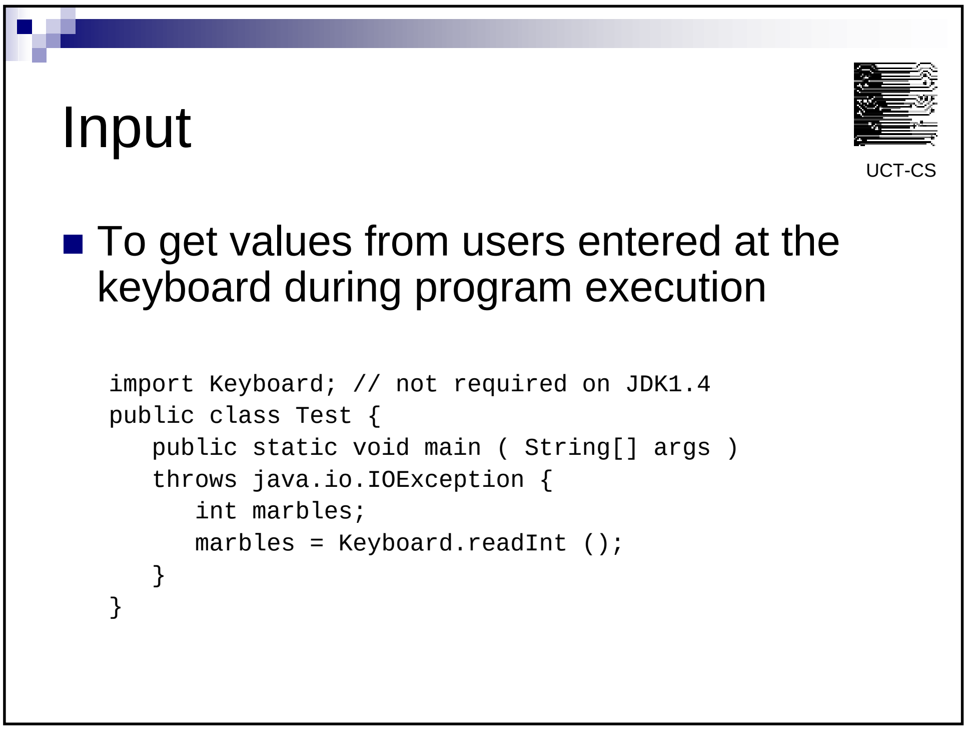
OOP: Methods: Quick Quiz

UCT-CS

```
public class Planet {
   private String name;
   public void setName ( String aName ) {
      name = aName;
    }
}
...
Planet earth = new Planet ();
```

- Which of these work?

```
earth.setName ();
earth.setName (2.345);
earth.setName (“Mars”);
earth.setName (“Mercury”, “Venus”, “Earth”);
earth.setName (“The”+“ Dude’s ”+“Planet”);
```

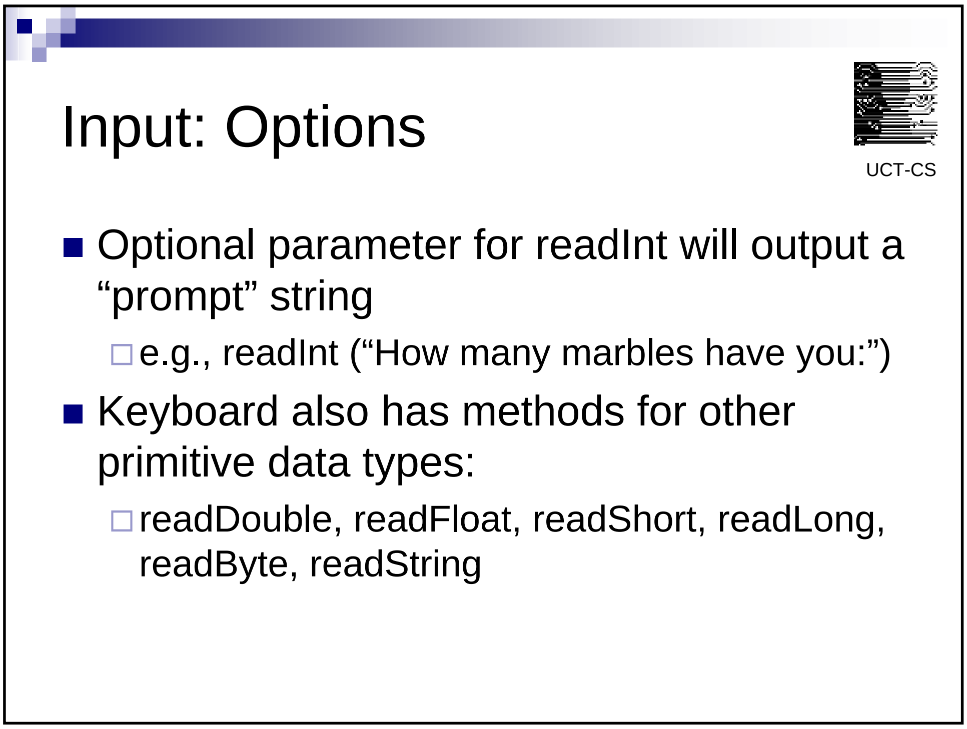


Input

UCT-CS

- To get values from users entered at the keyboard during program execution

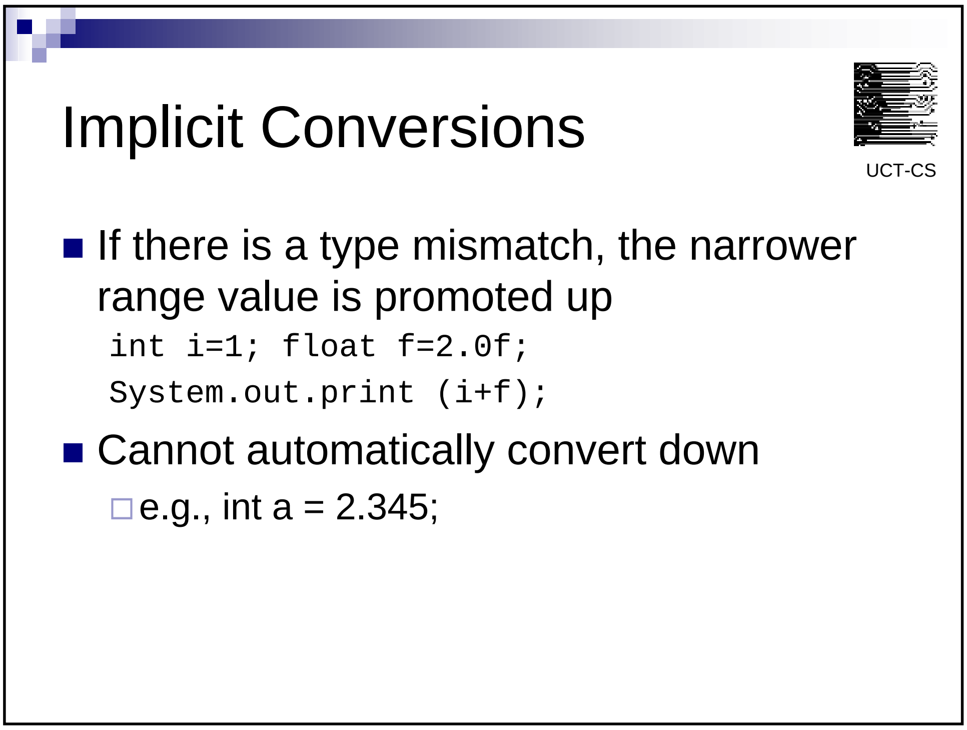
```
import Keyboard; // not required on JDK1.4
public class Test {
   public static void main ( String[] args )
   throws java.io.IOException {
      int marbles;
      marbles = Keyboard.readInt ();
   }
}
```



Input: Options

UCT-CS

- Optional parameter for readInt will output a “prompt” string
  - e.g., readInt (“How many marbles have you:”)
- Keyboard also has methods for other primitive data types:
  - readDouble, readFloat, readShort, readLong, readByte, readString



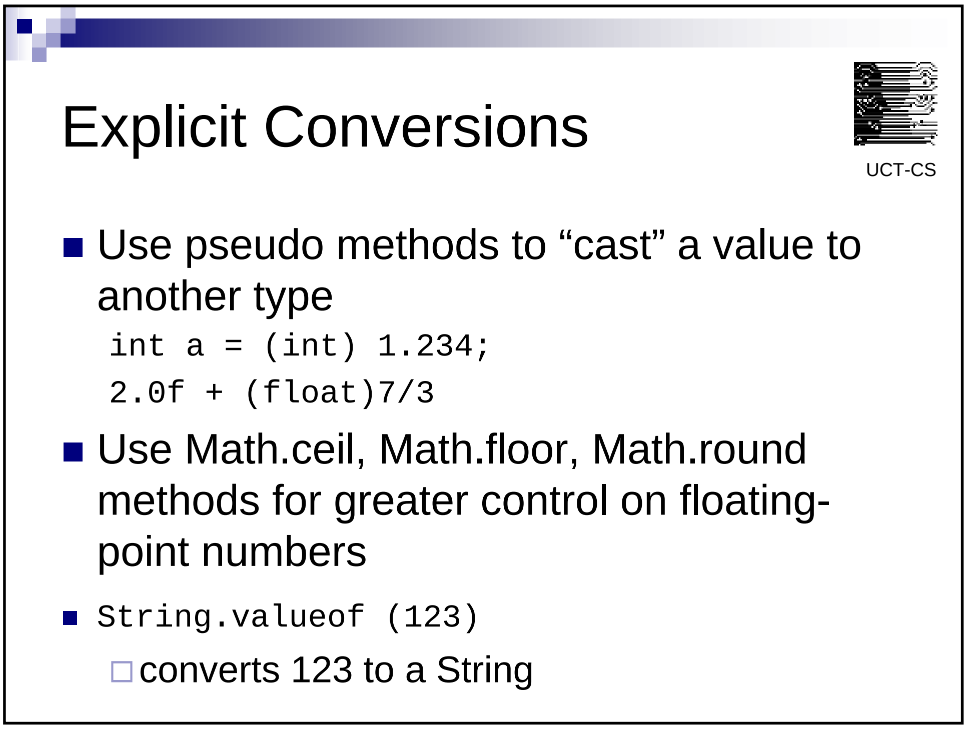
Implicit Conversions

UCT-CS

- If there is a type mismatch, the narrower range value is promoted up

```
int i=1; float f=2.0f;
System.out.print (i+f);
```

- Cannot automatically convert down
  - □ e.g., int a = 2.345;



Explicit Conversions

UCT-CS

- Use pseudo methods to “cast” a value to another type

```
int a = (int) 1.234;
2.0f + (float)7/3
```

- Use Math.ceil, Math.floor, Math.round methods for greater control on floating-point numbers
- `String.valueof (123)`
  - □ converts 123 to a String