TOPIC 2
INTRODUCTION TO JAVA AND DR JAVA

Notes adapted from Introduction to Computing and Programming with Java: A Multimedia Approach by M. Guzdial and B. Ericson, and instructor materials prepared by B. Ericson.

Outline

- DrJava
- Memory and Variables
- Types
- Boolean expressions
- Strings
- Java statements
- Variables
- Constants
- Objects
- References variables
- Naming conventions

Dr Java

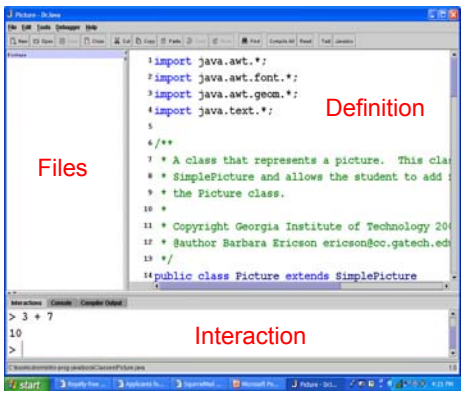
Where you code.

What is DrJava?

DrJava is an **IDE**

Integrated Development Environment for Java Programming

- It has several panes
  - Definitions pane
  - Interactions pane
  - Files pane



Interaction Window (pane)

- Where you can **interact** with code
- You can **practice** here
- To actually write code, you need certain "key words" and brackets surrounding the code
  - Here you don't need to know how to use those "key words" and can try writing bits of code on your own
- This does NOT work in the "real world", this is a feature of DrJava

Definitions Window (pane)

- Used for creating (typing in, editing) **complete** Java programs
- Need to use the "key words" and brackets to make it work (more on this later)
- This is how you write **real code**!!!
- You will use this when creating complete programs in your Labs, and for your assignments

•7

Memory and Variables

The beginning.

Memory

- In the computer there are places where things can be stored - “memory”
- You can put any “thing” you want in memory, but you must tell the computer how to interpret it
- For example, if you place a number in a slot in memory, you have to tell the computer it is a number so it knows how to handle it

Variables

- When you place something in memory to be used later, it is a **variable**
- For example if you want to add two numbers together, you would tell the computer to store the first number in some slot in memory, and tell it it is a number
- You would do the same with the second, then add them
  - Int number1 = 12;
  - Int number 2 = 10;
  - Number1 + number2;

More on this later! :)

Arithmetic expressions

Do some math.

Definition

- To try out DrJava, we will begin with simple math
- An arithmetic expression consists of operands (values) and operators (+ - * / %), and represents a numeric value
- **Examples**
  (3 + 4) * 5 – 67 / 8
  3.141592 * 5.0 * 5.0

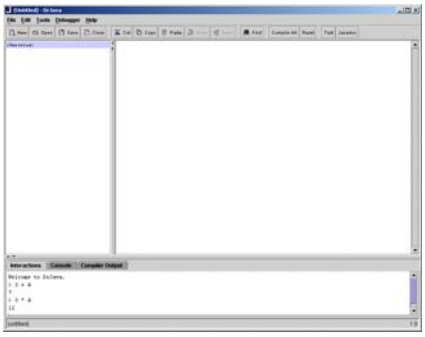
List of math operators

- Addition
  3 + 2 → 5
- Subtraction
  3 – 2 → 1
- Multiplication
  3 * 2 → 6
- Division
  3 / 2 → 1
- Negation
  -2
- **Modulo** (Remainder on Integer Division) *
  10 % 2 → 0

  11 % 2 → 1

Sample exercise

- In DrJava, do the following in the Interactions pane:
  - subtract 7 from 9
  - add 7 to 3
  - divide 3 by 2
  - multiply 5 by 10
  - find the remainder when 10 is divided by 3



Math operator order

- Default evaluation order is
  - parentheses
  - negation
  - multiplication, division, and modulo (remainder), from left to right
  - addition and subtraction, from left to right

- Examples:

  (3 + 4) * 2 versus 3 + 4 * 2

- We can use parentheses for readability: 3 + (4 * 2)

Sample exercise

- Try evaluating the expression 2 + 3 * 4 + 5
- Add parentheses to make it clear what is happening
- How do you change it so that 2 + 3 is evaluated first?
- How do you change it so that it multiplies the result of 2 + 3 and the result of 4 + 5?

The notion of type

3/2=1

3.0/2=1.5

3/2 = 1

- Java is what is a "**strongly typed language**"
- Each value has a **type** associated with it
- This tells the computer how to interpret a number:
  - integers are of type **int**
  - numbers with decimal points are called **floating-point numbers** and are of type **float** or **double**
  - **ints do not have decimals!**

3/2 = 1

- Recall in the "memory and variables" section we learned that we could store values in memory if we told the computer what it was
- This means we must give the computer the "type"
- We just saw the types integer and float
- What type did we use on slide 9?

3/2 = 1

- The Java compiler can determine the type of a number, for example:
  - 3 is an integer
  - 3.0 is a floating point number
- Rule: the result of an operation is the same type as the operands
  - 3 and 2 are integers, so the operation / is integer division, and the answer is the integer 1
- What is the result of 3.0 / 2.0 ?
  - What is the operation / here?

Type conversion

- What happens if you divide 3.0/2 ?
- Rule: If the types of the operands differ, Java automatically converts the integer to a floating point number
  - Why not the other way around?
- How else could you do the division 3/2 so that the result is 1.5?

Casting

- You can do the type conversion yourself: this is called **casting**
  - You cast an int value to a float or double, by putting that type in parentheses before the integer you want to have converted
  - Examples:
    - Cast 3 to a double: (double) 3 / 2
    - Cast 2 to a double: 3 / (double) 2

Sample exercise

- Use casting to get the values right for a temperature conversion from Fahrenheit to Celsius
  - Celsius is 5/9 * (Fahrenheit – 32)
- Try it first with a calculator
- Try it in DrJava without casting
- Try it in DrJava with casting

Try this at home!!!

•23

Primitive data types

Integer, floating-point, characters, booleans

Data types in Java

•24

In Java, there are two kinds of data types:

- **Primitive** data types
  - Used to store **simple data** values such as integers, floats, doubles, characters in main memory
  - Mainly for efficiency reasons
    - They take up little room in memory and allow fast computation
- **Reference** data types
  - Used to refer to objects (more on this soon)

Java primitive data types

- **Integers**
  - types: **int** or byte or short or long
  - examples: 235, -2, 33992093
- **Floating point numbers**
  - types: **double** (15 digits) or float (7 digits)
  - examples: 3.233038983, -423.9
  - called "floating point" because they are stored in scientific notation, for example:
    52.202 is 0.52202×10^2

Java primitive data types

- **Characters**
  - type: char
  - examples: 'a', 'b', 'A', '?'
- **Boolean** (true and false)
  - type: boolean
  - examples: true, false (the only possible boolean values)

Why so many different types?

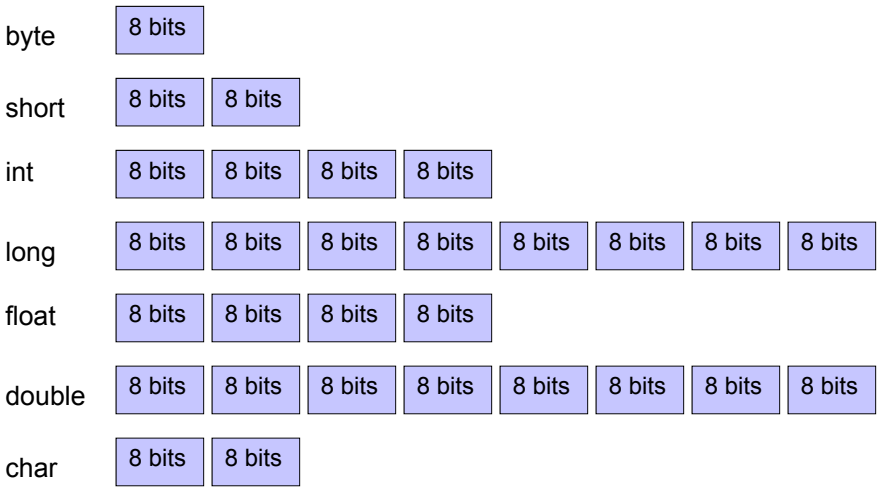
- They take up different amounts of space in memory
- Because the computer needs to know what to DO with them
- Numeric values have different **precisions**
  - integer values:
    - byte uses 8 bits (1 byte)
    - short uses 16 bits (2 bytes)
    - int uses 32 bits (4 bytes) (we usually use this)
    - long uses 64 bits (8 bytes)
  - floating point values:
    - float uses 32 bits (4 bytes)
    - double uses 64 bits (8 bytes) (we usually use this)

Why so many different types?

- A character (type char) is stored in 16 bits, in Unicode format (because computers only understand numbers)
  - Unicode is an industry standard encoding for characters
  - Examples:

Character	Encoding
A	65
a	97
{	123
1	49

Sizes of primitive types



Boolean expressions

Expressions that represent true or false

List of relational operators

- Greater than >
  - 4 > 3 is true
  - 3 > 3 is false
  - 3 > 4 is false
- Less than <
  - 2 < 3 is true
  - 3 < 2 is false
- **Equal** ==
  - 3 == 3 is true
  - 3 == 4 is false
- Not equal !=
  - 3 != 4 is true
  - 3 != 3 is false
- Greater than or equal >=
  - 4 >= 3 is true
  - 3 >= 3 is true
  - 2 >= 4 is false
- Less than or equal <=
  - 2 <= 3 is true
  - 2 <= 2 is true
  - 4 <= 2 is false

Relational operators

- **Relational operators** compare two operands of the same type
- The result of the operation is either true or false
  - So the result is of type boolean
- The symbol for equality is ==

  (we will see later that = is used for something else)

Sample exercise

- Try out **relational expressions** in the Interactions pane
  - With numbers

    3 < 4

    4 <= 4

    5 < 4

    6 == 6.0 (what is the reason for the result?)
  - With characters (use single alphabet letters)

    Rule: Put **single** quotes around a character

    'a' < 'b'

    'b' < 'a'

    'a' == 'a'

    'a' == 'A'

Strings

Strings are not a primitive data type!

Strings in Java

- A **string** is a **sequence of characters**, for example
  Programming is fun!
- **Text data** is represented in computers as a string, i.e. a sequence of characters in Unicode
  - Example: The string CS1026 is represented by the sequence of codes

    67 83 49 48 50 54
- Java has a type called String for string data
  - In Java a string is an **object**, so String is **not** a primitive type

Strings in Java

- The Java compiler recognizes strings as beginning and ending with " (a double quote)
- Rule: put double quotes around a string
- A string can have many characters, for example:
  "This is one long string with spaces in it."
- A string can have no characters
  - This is called the the **null string**
  - It is represented by ""
    (double quotes with nothing between)

Strings in Java

- Java can add (or **concatenate**) strings to other strings, using the **concatenation operator +**
  - This returns a new string with the characters of the second string appended after the characters of the first string
  - Examples: what strings are formed by the **string expressions**

```
"CS1026" + "a"                 becomes CS1026a
"CS1026" + "b"                 becomes CS1026b
"CS1026" + "a" + "/" + "b"     becomes CS1026a/b
```

Strings in Java

- Now you see why it is important to tell the computer the type you have stored in memory
- If you just stored 2 strings and didn't tell the computer they were strings, and it thought they were numbers,

"CS1026" + "a"

Would give you a very different result than you were looking for! An error would pop out!

Strings in Java

- There is a special character \ in Java strings called the **escape character**
- It is used to allow **special characters** to be embedded into a string
  - Examples:
    - \" Used to put a " into a string
    - \\ Used to put a \ into a string
    - \t Used to put a tab into a string
    - \n Used to force a new line in a string

Sample exercise

- How would you print the following on the console with a single println statement?

```
Course name is "CS026"
Directory is "koala\Homes\Students"
```

Try this at home – it is harder than it sounds!

Java statements

The example of System.out.println

Statements

- Java programs are made up of statements
  - Like sentences in English
  - But Java statements end in a semicolon, not a period
- Missing semicolons in a Java program lead to a lot of syntax errors!
- Examples of Java statements:

```
System.out.println(3*28);
numPeople = 2;      (an assignment statement)
```

Printing

- We often want to output the value of something in a program
- In Java, we print to the screen using

  `System.out.println(expression);`
  - To print the value of the expression in the parentheses, and then go to a new line

  `System.out.print(expression);`
  - To print just the expression in the parentheses without a new line afterwards
- These are **Java statements**.

Sample printing exercise

- Use `System.out.println()` to print the results of an expression to the console:

```
System.out.println(3 * 28);
System.out.println(14 – 7);
System.out.println(10 / 2);
System.out.println(128 + 234);
```

- Try using `System.out.print(…)` instead
  - What is the difference?

More on Variables

Variables

- We've used Java to do calculations and concatenate strings, but we haven't **stored** the results
- The results are in memory somewhere, but we don't know where they are, and we don't know how to get them back
- To solve this problem, we use variables

Variables

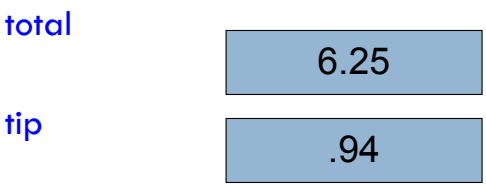
- **Variables** are locations in memory containing a value, labeled with a name
- They are called "variables" because their contents can vary – recall, we need to tell the computer what the type is!
  - We can store data in a variable
  - We can perform a calculation and store the results in a variable
- We access stored values by using their variable names

Variables

- Suppose the variable total represents your total bill at a coffee shop, and its contents was the result of adding the price of coffee and a sandwich. If you wanted to calculate what a tip of 15% would be, you could do this using the expression

  total * .15

  and storing that in a variable called tip



Variables in Java

- In Java programs, variables are created and named by **declaring** them
- To **declare a variable** you specify a type for the variable, and give it a name
  - Providing a type lets Java know how much memory to set aside for the variable, and what operations can be done on that variable
  - Choose meaningful variable names so that it is easier to write and read your program
- You **must** declare a variable before you use it

Variable declarations

- In general, variables are **declared** like this: type name;
- type is a special "keyword" in Java and there are only a few; name is something you pick (though there are some rules)
- **Example**: we have several people in a restaurant, and we want to know how much each should pay, including the tip. We'll start by declaring some variables:

```
int numPeople;
double bill, tip;
```

- Three variables: one integer variable (numPeople) and two floating point variables (bill and tip)
  - Java allows multiple variables to be declared at the same time

Assignments

- Values are stored to variables in Java using **assignment statements**

  name = expression;

  - This is a Java statement, **so it ends with a semicolon**
  - We read **=** as assigning the value from the expression on the right side to the variable named on the left
- Our restaurant example:

  numPeople = 2;

  This assigns the value 2 to the integer variable numPeople that we declared earlier

Storing values in variables

- One can declare variables and assign initial values to them at the same time
- Example: we can combine the declaration of a variable with an assignment:

  int numPeople = 2;

  double bill = 32.45;

Using variables

- A variable name can be used wherever a constant value of the same type could be used, but
  - The variable must be declared first

    Our example: bill and tip have already been declared

    double total = bill + tip;
  - The variable must have been assigned a value first
    - Why? to ensure that it has a valid value stored in it
    - Our example: bill and tip have already been declared and initialized

Example: using variables


```
int numPeople = 2;
double bill = 32.45;
double tip = bill * 0.20;
double total = bill + tip;
double totalPerPerson = total / numPeople;
System.out.println("You each pay " +
        totalPerPerson);
```

An equivalent form


```
int numPeople;
double bill, tip, total, totalPerPerson;

numPeople = 2;
bill = 32.45;
tip = bill * 0.20;
total = bill + tip;
totalPerPerson = total / numPeople;
System.out.println("You each pay " +  totalPerPerson);
```

Variable declarations revisited

- Recall that declaring a variable creates and names it
- By default, Java also initializes a variable to a **default value**
  - 0 for variables of type int
  - 0.0 for variables of type float and double
- Example: what are these variables initialized to?

```
int numPeople;
double bill, tip;
```

Tracing through code

- It is often useful to **trace through code** to see what it does
  - When we are debugging a program we have written, or trying to understand a program someone else has written
- A good way to do this:
  - Draw and label boxes for each variable in the code
  - Follow through the code, making changes to the variables

numPeople	2	tip	6.49
bill	32.45	total	38.94

Constants

Magic numbers

- In our restaurant example, we used a **constant value** (aka **literal**) for the tip percentage:

  0.20

- We call this kind of constant a "**magic number**"
  - Why? The constant means something to the programmer, but maybe means nothing to someone else reading the code
- Using **magic numbers** in programming is considered a poor practice, as it makes code difficult to read and change

Named constants

- We should use **named constants** that have meaningful names given to them
- To create a named constant:
  - declare a variable and assign a value to it
  - add the keyword **final** to signify that this variable is a named constant
- Example:

```
final double TIP_RATE = 0.20;
double tip = bill * TIP_RATE;
```

- **Naming convention (rule)**: named constants are in all uppercase letters, with underscores separating words, for example TIP_RATE

Common errors

Common errors

- Simple typos
  - The most common error!
  - Examples of errors:

```
duble total = bill + tip;
double total = bil + tip;
```

- Case sensitivity
  - Java is **case sensitive** and will generally treat issues with case as it would typos
  - Examples of errors:

```
doublE total = bill + tip;
double total = Bill + tip;
```

Common errors

- Redeclaring a variable
  - Once a variable has been declared, you should not declare another variable with the same name
  - Example of error:

```
int numPeople = 2;
…
int numPeople;
```

- Reassigning a constant
  - A constant's value cannot be changed!
  - Example of error:

```
final double TIP_RATE = 0.20;
TIP_RATE = 0.15;
```

Common errors

- Loss of precision
  - Java will automatically convert integer values to floating point values as necessary
  - Java will **not** automatically convert floating point values to integers, as this could result in a loss of precision … you must cast instead
  - Example of error: `int age = 5.6;`
- Uninitialized variables
  - Java usually wants variables to be initialized

  before they are used
  - Example of error:

```
int bill, tip;
tip = bill * 0.20;
```

Objects in Java

Consider a Problem...

- Suppose that we were interested in trying to use the computer to model some illness. We would need to talk about the "pieces":
  - There was the body, a cell, an organ, etc.
- When we want represent these in a program, we find ways to represent them - they are called "**Objects**" → makes sense, right?
- They represent real life things and we can give them properties

Our Problem...

- A body would have an age, or a cell might have a size? These are **attributes**/**properties** of the object
- We can talk about what the objects might do? Like a cell might divide, or a protein might attach to a cell. Those are **actions**!
- We are going to learn programming in a way that is oriented towards Objects!

Object oriented?

- **Objects** are
  - **persons, places, or things** that can do actions or be acted upon in a Java program
- Objects have
  - **Properties**
  - **Actions**
- Every object belongs to a specific **class**
  - Objects that belong to the same class have the same kinds of properties and behaviors

Back to example

- So, objects that belong to the same class have the same properties and behaviors
- We have a class called "Cell"
- Any Cell objects will have the same properties → they will have a size, an age, a shape perhaps
- If we have a class "Body" each body will have the same properties
- I could make 3 bodies: Jenna, Joe, Bob
- Each would have an age, a name, a height, a weight… they might have different names or weights or ages or heights, but they have the same attributes!

Let's talk about a restaurant!

Another Example

- In a **restaurant:**
  - When customers enter the restaurant, a greeter welcomes them and seats them at a table
  - A waiter takes their order, and one or more chefs prepare the order
  - The waiter brings the drinks and food, and when the customers are done, the waiter creates and brings them their bill
  - On their way out, the customers pay the bill

Example

- Each of the entities involved in this scenario is an ***object***
- The **objects** in this scenario worked together to get the job done (feeding the customers)

Chefs

- Suppose that there were two chefs working on the order, Alice and Bob
- We will call Chef a **class**, with both Alice and Bob being **objects** that belong to the class Chef
- Alice and Bob are **instances of the class** Chef

Common properties and actions

- Like all chefs, Alice and Bob have common properties and actions
  - Properties: they both have a name, a set of dishes they know how to prepare, a number of years of experience, and so on
  - Actions: they both are able to talk, prepare ingredients, cook dishes, and so on

Specificities

- Even though they are both chefs, they are also still individuals
  - They have their own set of the properties of the class Chef: name, dishes they can cook, etc.

Summary

- Objects and classes in a Java program work similarly to these examples:
  - All the objects work together to get the task done, even though each object plays a different role
  - Each object belongs to a class
    - All objects in the same class have the same kinds of properties and behaviours
    - But all objects of the class are still distinct entities
- We will be seeing a lot more on objects

Reference variables

How to refer to an **object**.

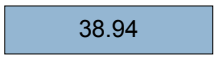
Reference variables

- **Simple variables**: All the variables we have discussed so far have been for storing values of primitive types
- **Reference variables (object variables)** are variables that are used to refer to objects
  - They do not store the objects themselves
  - Instead, they store the location of the objects so that they can be found when necessary
  - That's why we say they **refer to** objects, and call them reference variables

Reference variables

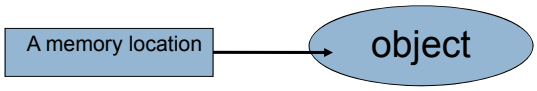
- We can think of a variable as being a little box in memory containing a value, labeled with its name
  - A simple variable contains a value
  - A reference variable contains the location of an object
- For a **simple** variable, we can draw it like this:

total



- For a **reference** variable, we can draw it like:

test

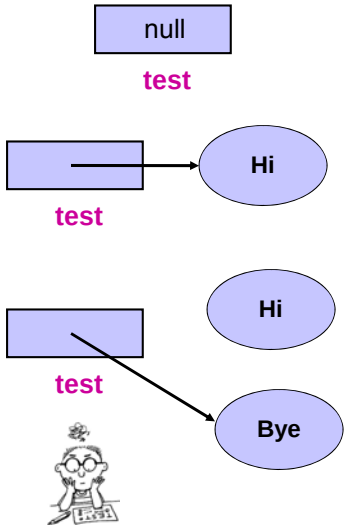


Reference variables example


```
String test;
System.out.println(test);

test = "Hi";
System.out.println(test);

test = "Bye";
System.out.println(test);
```



Using reference variables

- Reference variables are declared in general by:

  type **name;**

- Example: Strings are **objects.** Declare a reference variable that will refer to a string:

  String **test;**

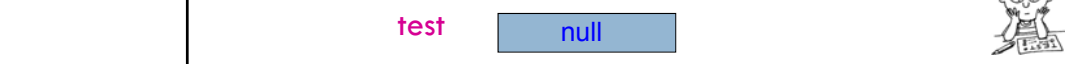
  - This does not create a String object
  - It only declares **a variable named test that can refer to a String object**
  - By default, it does not refer to anything yet, and is said to be a **null reference**

The null reference

- Java has a keyword **null** that means the null reference
- Example: the declaration

  String test;

  by default stores null in the variable test



- This is the same as if we had declared and initialized this variable by

  String test = null;

Using reference variables

- To have a reference variable refer to an object, we use an assignment statement
  - Example:

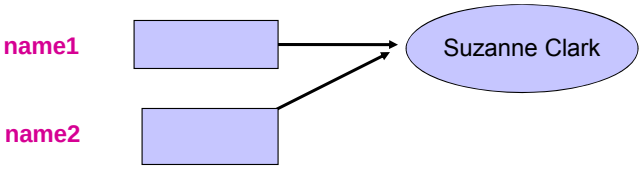
    test = "Hi";
    - Java will create a String object containing the text Hi and have the variable test refer to it

    test = "Bye";
    - Java will create another String object containing the text Bye and have test now refer to it
  - We will learn more about Strings later on

Multiple references to objects

- In Java, it is possible to have multiple references to the same object! Consider:

```
String name1 = "Suzanne Clark";
String name2 = name1;
```

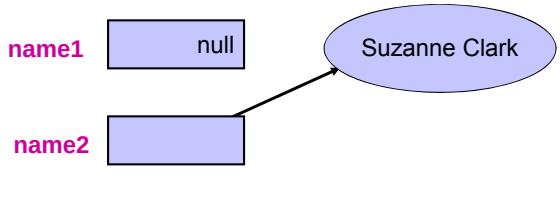


- In this case, name1 and name2 refer to the same object in memory
- This is called **identity equality:** the two variables have the same contents

Multiple references to objects

- Important note: the two references are independent, and what you do to one does not affect the other
- Following the example on the previous slide, now consider this:

```
name1 = null;
```

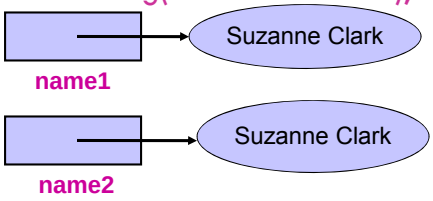


  - This change only affects the reference variable name1, and not name2

Multiple references to objects

- Let's create two objects (the "new" operator is used to create new objects – more on that later):

```
String name1 = "Suzanne Clark";
String name2 = new String("Suzanne Clark");
```



  - In this case, name1 and name2 refer to two different objects, but with the same contents
  - This is called **state equality:** the two variables refer to objects with the same contents

Naming conventions

Variable declarations

- Java has a variable **naming convention**:
  Variable names start with lowercase letters, but uppercase the first letter of each additional word
- Examples:
  - bill
  - tip
  - numPeople

Java naming conventions

- Class names start with an uppercase letter, for example:

  System, String, Picture

- Named constants are in all uppercase letters, with underscores separating words, for example:

  TIP_RATE

- All other names start with lowercase letters, but uppercase the first letter of each additional word, for example:

  picture, fileName, thisIsALongName

Java naming conventions

- Java code will compile if you don't follow these conventions, but it may be hard for other programmers to understand your code
- As an example, can you identify which of these are primitive types, and which are the names of classes, just by following conventions?
  - char
  - Double
  - Math
  - double
  - Integer
  - String

Summary of java concepts

- Objects, Classes
- Object properties, behaviours
- Math operators
- Primitive types
- Casting
- Printing output
- Relational operators
- Strings
- Variables
- Assignment statements
- Named constants
- References to objects
- Naming conventions

Key Notes

- Modulo → practice at home
- Order of operations
  - Do these evaluate to the same answer?
  - (2 * 3)+1 and
  - 2 * 3 + 1
- Int division vs double division
- Always put a semi colon after a Java statement
- Practice naming conventions (you are graded on this during assignments)