NUS National University of Singapore

School of Computing

CS2040 Data Structures and Algorithms

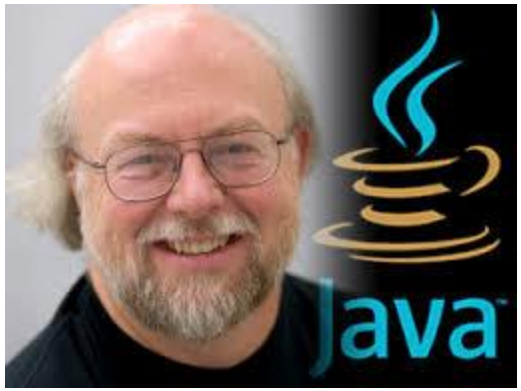
Lecture Note #1

Introduction to Java

Outline

1. Java: Brief history and background
2. Run cycle
3. Basic program structure
4. Basic Java program elements
   - 4.1 Primitive and Reference Types
   - 4.2 Control Flow Statements and Logical Expressions
   - 4.3 Basic Input (Scanner class) and Output
   - 4.4 User defined method (class method)
   - 4.5 Useful Java API classes – Scanner, Math, String
   - 4.6 Essential OOP concepts for CS2040

1. Java: Brief History & Background



James Gosling
1995, Sun Microsystems

Use C/C++ as foundation
- "Cleaner" in syntax
- Less low-level machine interaction

- Write Once, Run Everywhere™
- Extensive and well documented standard library

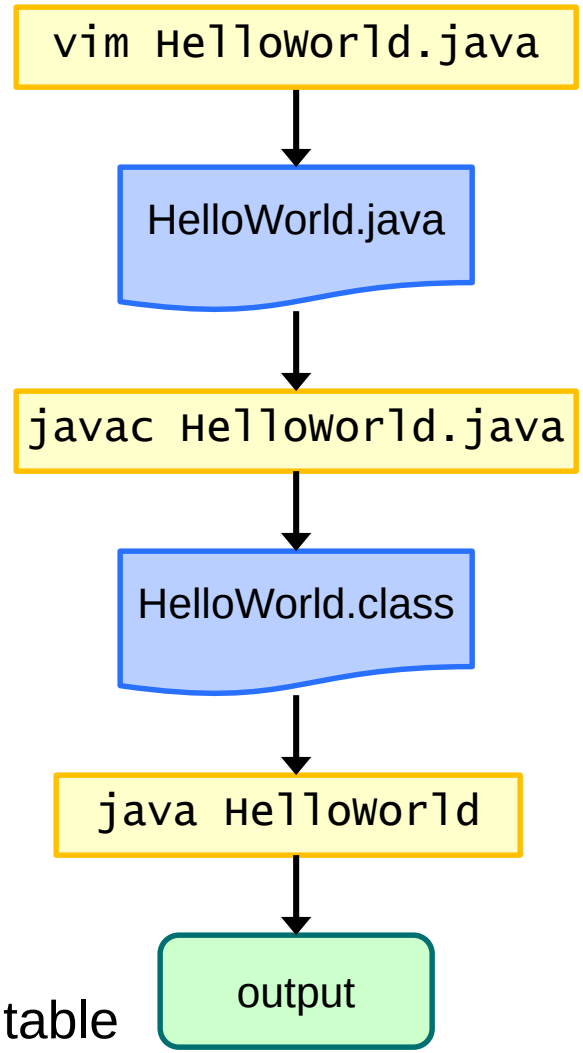
- Less efficient

Java: Compile Once, Run Anywhere?

- Normal executable files are tied to OS/Hardware
  - An executable file is usually not executable on different platforms
  - E.g: The a.out file compiled on sunfire is not executable on your Windows computer
- Java overcomes this by running the executable on an uniform hardware environment simulated by software
  - This is the Java Virtual Machine (JVM)
  - Only need a specific JVM for a particular platform to execute all Java bytecodes without recompilation

Run Cycle for Java Programs

- Writing/Editing Program
  - Use an text editor, e.g: vim
  - File must have `.java` extension
- Compiling Program
  - Use a Java compiler, e.g.: **`javac`**
  - Compiled binary has `.class` extension
  - The binary is also known as **Java Executable Bytecode**
- Executing Binary
  - Run on a ***Java Virtual Machine (JVM)***
    - e.g.: `java HelloWorld` (leave out the **.class** extension)
  - Note the difference compared to C executable



Hello World!


```c
#include <stdio.h>

int main(void) {
    printf("Hello World!\n");

    return 0;
}
```

HelloWorld.c


```java
import java.lang.*;  // optional

public class HelloWorld {

  public static void main(String[] args) {

     System.out.println("Hello World!");

  }
}
```

HelloWorld.java

Hello World! - Dissection (1/3)

```
import java.lang.*;  // optional
public class HelloWorld {
  public static void main(String[] args) {
     System.out.println("Hello World!");
  }
}
```

HelloWorld.java

- The main() method (function) is enclosed in a "**class**"
- There may be multiple classes in a program
- There can be one and only one **public** class and it is the one containing the main() method, which serves as the starting point for the execution of the program
- Each class will be compiled into a separate **`XXX.class`** bytecode
  - "**`XXX`**" is taken from the class name ("**`HelloWorld`**" in this example)

Hello World! - Dissection (2/3)

```
import java.lang.*;  // optional

public class HelloWorld {

   public static void main(String[] args) {

      System.out.println("Hello World!");

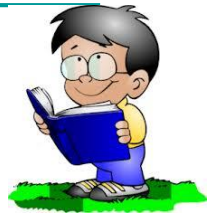
   }
}
```

HelloWorld.java

Beginners' common mistake:
Public class name not identical to program's file name.

- File name must be the same as the public class name

Hello World! - Dissection (3/3)



```
import java.lang.*;  // optional
public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello World!");
   }
}
```

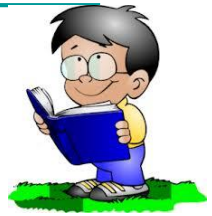
HelloWorld.java

- To use a predefined library in Java have to import it
  - Using the "`import XXXXXX`;" statement
- A library in Java is known as a **package**
- Packages are organized into hierarchical grouping
  - E.g "**`System.out.println()`**" is defined in "**`java.lang.System`**", i.e. "**`lang`**" is a package under "**`java`**" (the main category) and "**`System`**" is a class under "**`lang`**"
- All packages/classes under a group can be imported with a "**`*`**"
- Packages under "**`java.lang`**" are imported **by default**

4 Basic Program Elements

4.1 Primitive and Reference Types

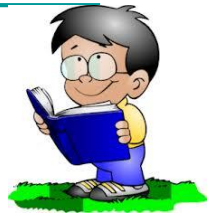
Identifier, Variable, Constant (1/2)



- **Identifier** is a **name** that we associate with some program entity (class name, variable name, parameter name, etc.)
- Java Identifier Rule:
  - May consist of letters ('a' – 'z', 'A' – 'Z'), digit characters ('0' – '9'), underscore (_) and dollar sign ($)
  - Cannot begin with a digit character
- **Variable** is used to store data in a program
  - A variable must be declared with a specific data type (*for statically-typed languages and Java is such a language*)
  - Eg:

```
int countDays;
double priceOfItem;
```

Identifier, Variable, Constant (2/2)

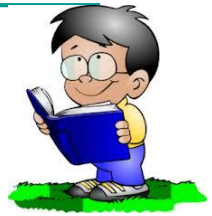


- **Constant** is used to represent a fixed value
  - Eg: `public static final int PASSING_MARK = 65;`
  - Keyword **`final`** indicates that the value cannot change
- Guidelines on naming
  - Class name: UpperCamelCase
    - Eg: **`Math`**, **`HelloWorld`**, **`ConvexGeometricShape`**
  - Variable name: LowerCamelCase
    - Eg: **`countDays`**, **`innerDiameter`**, **`numOfCoins`**
  - Constant: All uppercase with underscore
    - Eg: **`PI`**, **`CONVERSION_RATE`**, **`CM_PER_INCH`**
  - Reference from old module … → http://www.comp.nus.edu.sg/~cs1020/labs/styleguide/styleguide.html

Primitive and Reference Types

- Data types in Java are categorized into 2 groups
  - Primitive types – byte, short, int, long, float, double, char, boolean
    - Variable of primitive type "store" a value of the same type as the variable in the stack (fast memory access)
  - Reference types – any class in Java
    - Variable of reference type do not store a value of the same type as the reference but rather the "memory address" of a value (more accurately an object) of the reference type in the heap (slower memory access)
    - This memory address is also know as a reference/pointer
    - In order to create/instantiate an object of the reference type we have to use the "new" keyword (will see this later)

Primitive Numeric Data Types

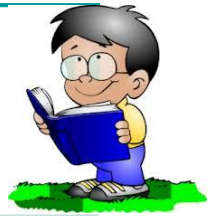


- Summary of numeric data types (primitive types) in Java:

	Type Name	Size (#bytes)	Range
Integer Data Types	`byte`	1	-2^7 to 2^7-1
	`short`	2	-2^{15} to $2^{15}-1$
	`int`	4	**-2^{31} to $2^{31}-1$**
	`long`	8	-2^{63} to $2^{63}-1$
	`char`	2	0 to $2^{16}-1$ (all unicode characters)
Floating-Point Data Types	`float`	4	Negative: -3.4028235E+38 to -1.4E-45 Positive: 1.4E-45 to 3.4028235E+38
	`double`	8	**Negative: -1.7976931348623157E+308 to -4.9E-324** **Positive: 4.9E-324 to 1.7976931348623157E+308**

- Usually you will use **`int`** for integers and **`double`** for floating-point numbers
- **`char`** can be considered an integer data type as each character is associated with an integer ASCII value

Numeric Operators



Higher Precedence

`()`	Parentheses Grouping	Left-to-right
`++, --`	Postfix incrementor/decrementor	Right-to-left
`++, --` `+, -`	Prefix incrementor/decrementor Unary +, -	Right-to-left
`*, /, %`	Multiplication, Division, Remainder of division	Left-to-right
`+, -`	Addition, Subtraction	Left-to-right
`=` `+= -= *= /= %=`	Assignment Operator Shorthand Operators	Right-to-left

- Evaluation of numeric expression:
  - Determine grouping using precedence
  - Use associativity to differentiate operators of same precedence
  - Data type conversion is performed for operands with different data type

Numeric Data Type Conversion

- When operands of an operation have differing types:
  1. If one of the operands is **`double`**, convert the other to **`double`**
  2. Otherwise, if one of them is **`float`**, convert the other to **`float`**
  3. Otherwise, if one of them is **`long`**, convert the other to **`long`**
  4. Otherwise, convert both into **`int`**

- When value is assigned to a variable of differing types:
  - **Widening (Promotion):**
    - Value has a smaller range compared to the variable
    - Converted automatically
  - **Narrowing (Demotion):**
    - Value has a **larger range** compared to the variable
    - **Explicit type casting is needed**

Data Type Conversion

- Conversion mistake:

```
double d;
int i;

i = 31415;
d = i / 10000;
```

Q: What is assigned to **d**?

Ans: 3

What's the mistake? How do you correct it?

- Type casting:

```
double d;
int i;

d = 3.14159;
i = (int) d; // i is assigned 3
```

Q: What is assigned to **i** if **d** contains 3.987 instead?

Ans: Still 3 (decimal part is truncated, not rounded)

The **(int) d** expression is known as **type casting**

Syntax:

```
(datatype) value
```

Effect:
value is converted explicitly to the data type stated if possible.

Reference Types: Wrapper Classes (1/2)

- Reference based counterparts of primitive data types
- Sometimes we need the reference equivalent of these primitive data types
- These are called wrapper classes – one wrapper class corresponding to each primitive data type

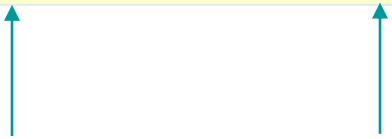
Primitive data type	Wrapper class
`int`	`Integer`
`long`	`Long`
`float`	`Float`
`double`	`Double`
`char`	`Character`
`boolean`	`Boolean`
and others…	

Instantiating/creating an object

- Since the wrapper classes are a reference type, after declaring a variable of the wrapper class type we still have to instantiate/create an object of the class to use it

- To instantiate/create an object of a class we need to use the **new** keyword eg

```
Integer i = new Integer(13);
```



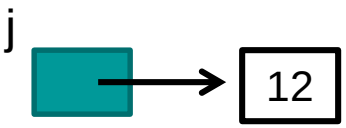
Declaration of an Integer reference variable

Instantiating an Integer object with the value 13 using the Integer constructor

Schematic view of primitive vs reference type in memory

- Eg int i = 7

- Eg Integer j = new Integer(12)



Reference Types: Wrapper Classes (2/2)

- We may convert a primitive type value to its corresponding object. Example: between **`int`** and **`Integer`**:
  - ```
    int x = 9;
    Integer y = new Integer(x);
    System.out.println("Value in y = " + y.intValue());
    ```
- Wrapper classes offer methods to perform conversion between types
- Example: conversion between string and integer:
  - **`int num = Integer.valueOf("28");`**
    - **`num`** contains 28 after the above statement
  - **`String str = Integer.toString(567);`**
    - **`str`** contains “567” after the above statement
- Look up the Java API documentation and explore the wrapper classes on your own
    ```

Autoboxing/unboxing (1/2)

- The following statement invokes autoboxing

```
Integer intObj = 7;
```

- An **`Integer`** object is expected on the RHS of the assignment statement, but 7 of primitive type **`int`** is accepted.
- ***Autoboxing*** is the automatic conversion that the Java compiler makes between the primitive types and their corresponding wrapper classes
  - The primitive value 7 is converted to an object of **`Integer`**
- The Java compiler applies autoboxing when a primitive value is:
  - Passed as a parameter to a method that expects an object of the corresponding wrapper class
  - Assigned to a variable of the corresponding wrapper class

Autoboxing/unboxing (2/2)

- Converting an object of a wrapper type (e.g.: `Integer`) to its corresponding primitive (e.g: `int`) value is called unboxing.
- The Java compiler applies unboxing when an object of a wrapper class is:
  - Passed as a parameter to a method that expects a value of the corresponding primitive type
  - Assigned to a variable of the corresponding primitive type

```
int i = new Integer(5); // unboxing
Integer intObj = 7;     // autoboxing
System.out.println("i = " + i);
System.out.println("intObj = " + intObj);
```

```
i = 5
intObj = 7
```

Arrays in Java (1)

- An array in Java is a reference type
- You need to “new” an array instantiate an array object
- An array can store either primitive values or objects (more precisely the references pointing to them)

Arrays in Java (2)

- Declaring an array is as follows

```
<type> [] identifier;
```

e.g float [] height;

declares an array of floating point values

- To declare multiple dimensional arrays simply include as many [] as there are dimensions

e.g a 2 dimensional floating point array

float [][] weight;

Arrays in Java (3)

- To instantiate a height array of size say 10 and a weight array of size 10x10

```
height = new float[10];
weight = new float[10][10];
```

or simply

```
float [] height = new float[10];
float [] weight = new float[10][10];
```

- To access and modify the value at a particular index in height say 3 or in weight at say 0,1 (can only do this after array object is created

```
height[3] = 10.2;
weight[0][1] = 1001.1;
```

Arrays in Java (4)

- To initialize a height array with 5 values, and a weight array with 2x2 values

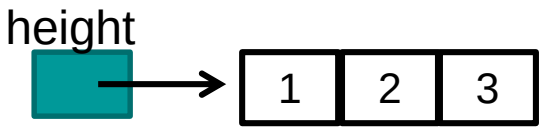
```
float [] height = {1.0,2.0,3.0,4.0,5.0}
float [][] weight = {{10.1,10.2},{10.3,10.4}}
```

1st row 2nd row

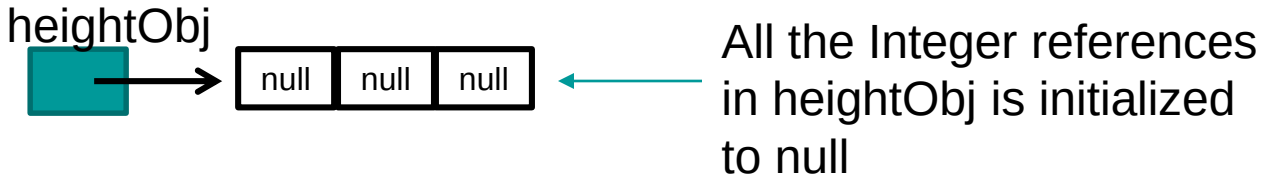
- Note that Java uses 0-based indexing

Schematic view of array in memory (1)

- E.g int [] height = {1,2,3}



- E.g Integer[] heightObj = new Integer[3];

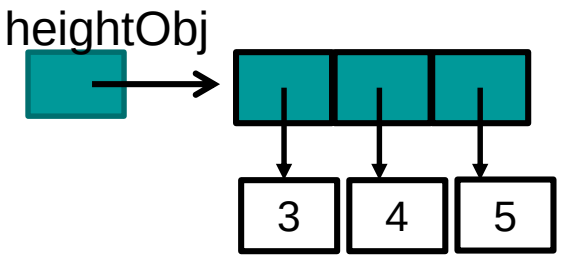


A reference type variable not pointing to any object is assigned null

Schematic view of array in memory (2)

- E.g

```
Integer[] height = new Integer[3];
height[0] = new Integer(3);
height[1] = new Integer(4);
height[2] = new Integer(5);
```



4.1 Problem: Adding 2 positive fractions

- Write a simple Java program FractionV1.Java:
  - Given 2 positive fractions $\frac{a}{b} + \frac{c}{d}$
  - Print out the resulting fraction from the addition
- For the time being, you can hard code the 2 fractions instead of reading it from user

4.1 Solution: Adding 2 fractions

FractionV1.java

```
public class FractionV1 {

  public static void main(String[] args) {
    int a,b,c,d,newNum,newDenom;

    a = 1;
    b = 2;
    c = 3;
    d = 4;
    newDenom = b*d;
    newNum = a*d+c*b;
    System.out.println("New Fraction = "+newNum+"/"+newDenom);
  }
}
```

Output:

```
New Fraction = 10/8
```

- Notes:
  - **10/8** is not the simplest form but it will suffice here
  - "**+**" in the printing statement
    - Concatenate operator, to combine strings into a single string
    - Variable values will be converted to string automatically
  - There is another printing statement, `System.out.print()`, which does not include newline at the end of line

4.2 Control Statements

Program Execution Flow

Boolean Data Type

- Java provides a **boolean** data type
  - Store boolean value ***true*** or ***false***, which are keywords in Java
  - Boolean expression evaluates to either ***true*** or ***false***

SYNTAX

```
boolean variable;
```

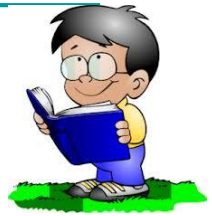
Example

```
boolean isEven;
int input;
// code to read input from user omitted
if (input % 2 == 0)
      isEven = true;
else
     isEven = false;
if (isEven)
      System.out.println("Input is even!");
```

Equivalent:

```
isEven = (input % 2 == 0);
```

Boolean Operators



	Operators	Description
Relational Operators	<	less than
	>	larger than
	<=	less than or equal
	>=	larger than or equal
	==	Equal
	!=	not equal
Logical Operators	&&	and
	||	or
	!	not
	^	exclusive-or

Operands are variables / values that can be compared directly.

Examples:

```
X < Y
1 >= 4
```

Operands are boolean variables/expressions.

Examples:

```
(X < Y) && (Y < Z)
(!isEven)
```

Selection Statements

```
if (a > b) {
    ...
}
else {
    ...
}
```

- `if-else` statement
  - else-part is optional
- Condition:
  - **Must be a *boolean* expression**
  - **Unlike C, integer values are NOT valid**

```
switch (a) {
    case 1:
        ...
        break;
    case 2:
    case 3:
        ...
    default:
}
```

- `switch-case` statement
- Expression in `switch()` must evaluate to a value of `char`, `byte`, `short` or `int` type
- `break`: stop the fall-through execution
- `default`: catch all unmatched cases; may be optional

Repetition Statements (1/2)

```
while (a > b) {
    ... //body
}
```

```
do {
    ... //body
} while (a > b);
```

- Valid conditions:
  - **Must be a *boolean* expression**
- `while` : check condition before executing body
- `do-while`: execute body before condition checking

```
for (A; B; C) {
    ... //body
}
```

- `A`: initialization (e.g. `i = 0`)
- `B`: condition (e.g. `i < 10`)
- `C`: update (e.g. `i++`)
- Any of the above can be empty
- Execution order:
  - `A, B, body, C, B, body, C, …`

4.3 Basic Input/Output

Interacting with the outside world

Reading input: The Scanner Class

PACKAGE

```
import java.util.Scanner;
```

SYNTAX

```
//Declaration of Scanner "variable"
Scanner scVar = new Scanner(System.in);

//Functionality provided
scVar.nextInt();
```

Read an integer value from source System.in

```
scVar.nextDouble();
```

Read a double value from source System.in

```
......
```

Other data types, to be covered later

Reading Input: Fraction Ver 2

```
import java.util.*;   // or import java.util.Scanner;

public class FractionV2 {

  public static void main(String[] args) {
    int a,b,c,d,newNum,newDenom;
    Scanner sc = new Scanner(System.in);

    System.out.print("Enter 2 Fractions to be added: ");
    a = sc.nextInt();
    b = sc.nextInt();
    c = sc.nextInt();
    d = sc.nextInt();

    newDenom = b*d;
    newNum = a*d+c*b;
    System.out.println("New Fraction = "+newNum+"/"+newDenom);
  }
}
```

FractionV2.java

Reading Input: Key Points (1/3)

```
import java.util.*;   // or import java.util.Scanner;

public class FractionV2 {

  public static void main(String[] args) {
     int a,b,c,d,newNum,newDenom;
     Scanner sc = new Scanner(System.in);

     //rest of code omitted
  }
}
```

FractionV2.java

- Declares a variable "**sc**" of **Scanner** type
- The initialization "**new Scanner(System.in)**"
  - Constructs a **Scanner** object (*discuss more about object later*)
  - Attaches it to the standard input "**System.in**" (the keyboard)
    - **sc** will receive input from this source
  - Scanner can attach to various input sources; this is one typical usage

Reading Input: Key Points (2/3)

```
import java.util.*;   // or import java.util.Scanner;

public class FractionV2 {

  public static void main(String[] args) {
    int a,b,c,d,newNum,newDenom;
    Scanner sc = new Scanner(System.in);

    System.out.print("Enter 2 Fractions to be added: ");
    a = sc.nextInt();

    // rest of code omitted
  }
}
```

FractionV2.java

- After proper initialization, scanner object provide functionality to read input from the input source
- ***nextInt()*** in ***sc.nextInt()*** works like a function (method in Java) that returns an integer value read interactively (in this case input from keyboard)
- In general the Scanner object **sc** converts the input into the appropriate data type and returns it

Reading Input: Key Points (3/3)

- Typically, only one Scanner object is needed, even if many input values are to be read.
  - The same Scanner object can be used to call the relevant methods to read input values
- Warning:
  - If you declare two or more Scanner objects in one program, you may get unexpected errors when you run your program. Especially when you submit your program to CodeCrunch.

Writing Output: The Standard Output

- **`System.out`** is the predefined output device
  - Refers to the monitor/screen of your computer

SYNTAX

```
//Functionality provided
System.out.print( output_string );

System.out.println( output_string );

System.out.printf( format_string, [items] );
```

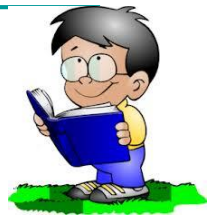
```
System.out.print("ABC");
System.out.println("DEF");
System.out.println("GHI");

System.out.printf("Very C-like %.3f\n", 3.14159);
```

Output:

```
ABCDEF
GHI
Very C-like 3.142
```

Writing Output: printf()



- Java introduces **`printf()`** in Java 1.5
  - Very similar to the C version
- The format string contains normal characters and a number of specifiers
  - Specifier starts with a percent sign (%)
  - Value of the appropriate type must be supplied for each specifier
- Common specifiers and modifiers:

`%d`	for integer value
`%f`	for double floating-point value
`%s`	for string
`%b`	for boolean value
`%c`	for character value

SYNTAX

`%[-][W].[P]``type`

`-`: For left alignment
`W`: For width
`P`: For precision

4.3 Problem: Add 2 fractions and output the new fraction in simplest form

- New requirement: Given the new fraction we want to express it in it's simplest form

  e.g 1: Simplest form of 2/3 is 2/3 itself

  e.g 2: Simpest form of 10/8 is 5/4

- Write FractionV3.java to:
  1. Ask the user for the 2 fractions to be added
  2. Calculate the new fraction
  3. Simplify the new fraction
  4. Output the new fraction

4.3 Solution: Using GCD

1. Compute GCD of numerator and denominator
2. Divide numerator and denominator by GCD
3. Output fraction with the new numerator and denominator

4.3 Solution

```
import java.util.*; // using * in import statement

public class FractionV3 {
  public static void main(String[] args) {
    // everything up to computation of newNum and newDenom as
    // in FractionV2

    int rem,e,f;

    e = newNum;
    f = newDenom;
    while (f > 0) {
      rem = e%f;
      e = f;
      f = rem;
    }  // GCD is the value of e after while loop stops
    newNum /= e;
    newDenom /= e;
    System.out.println("New Fraction = "+newNum+"/"+newDenom);
  }
}
```

FractionV3.java

4.4 User defined method (class method)

Reusable and independent code units

Writing a class method

- In FractionV3, we see that computing gcd is a useful function can be used in many mathematical problems
- In possible further extensions to our Fraction program, it is good not to have to keep re-writing that portion of code
- This can be achieved by writing a user defined method for gcd (like the System.out.println method) in Java called a **static/class method**
  - Denoted by the "`static`" keyword before return data type
  - Another type of method, known as **instance method** will be covered later

Writing a class method for gcd

FractionV4.java

```
public class FractionV4 {

  // Returns GCD of e and f
  // Pre-cond: e and f must be > 0
  public static int gcd(int e, int f) {
    int rem;
    while (f > 0) {
      rem = e%f;
      e = f;
      f = rem;
    }
    return e;
  }
  public static void main(String[] args) {

    // everything before computing gcd as in FractionV3

    int divisor = gcd(newNum,newDenom);
    newNum /= divisor;
    newDenom /= divisor;
    System.out.printf("New Fraction = "+newNum+"/"+newDenom);
  }
}
```

Method Parameter Passing

- All parameters in Java are passed by value (as in C/Python/Javascript (for primitives)):
  - A copy of the actual argument is created upon method invocation
  - The method parameter and its corresponding actual argument are two independent variables
- In order to let a method modify the actual argument, a **reference data type** is needed

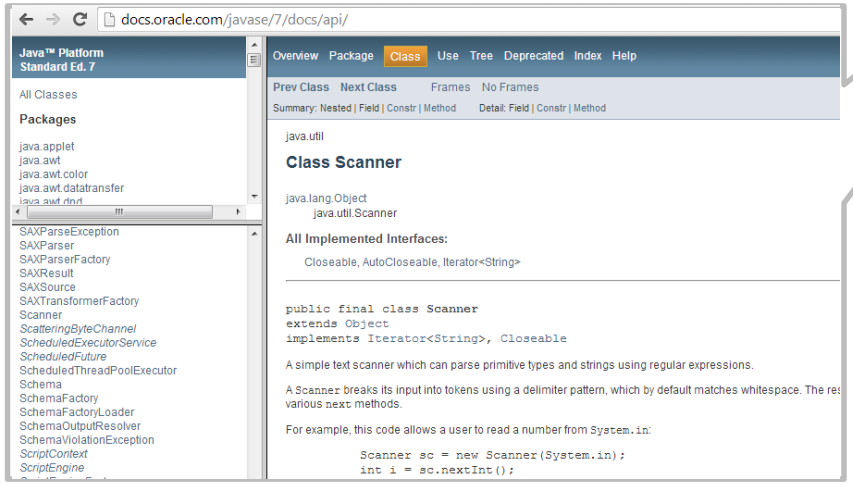
4.5 Useful Java API classes – Scanner, Math, String

Let’s look at the Java API (Application Programming Interface)

Java Programmer

API Specification

http://docs.oracle.com/javase/8/docs/api/



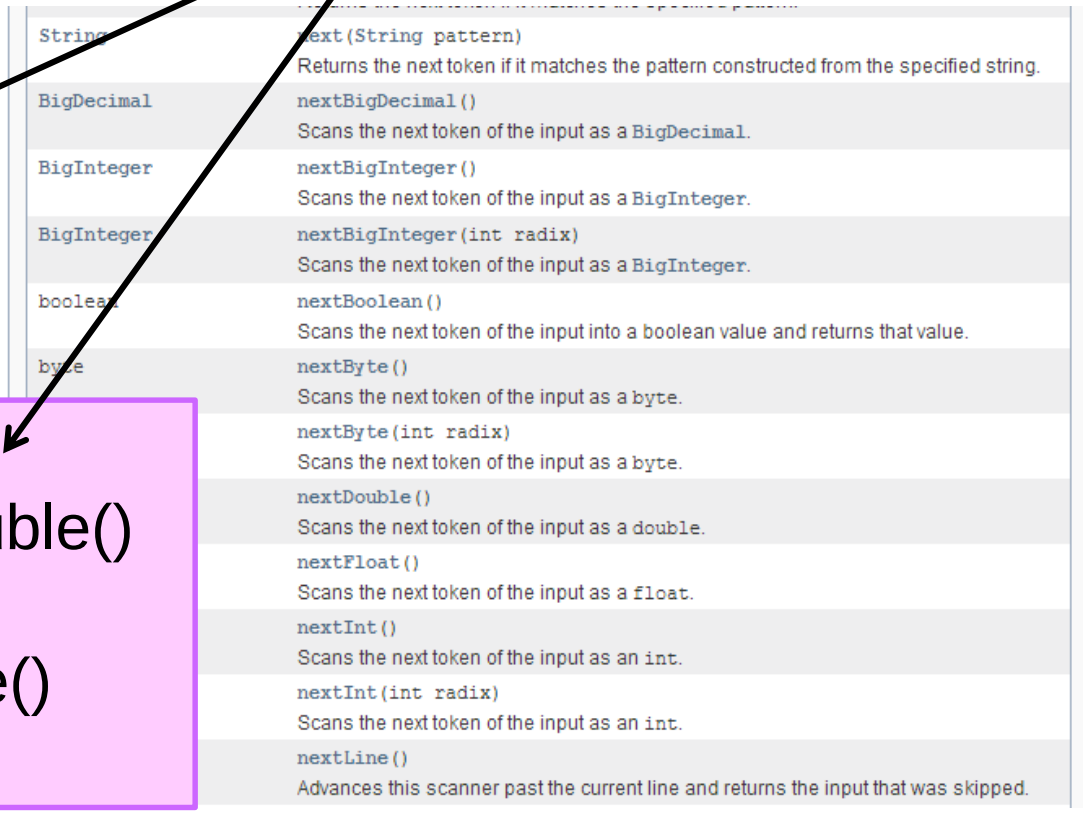
Scanner Class: Reading Inputs

- API documentation
  - http://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html
- For reading input
- Import java.util.Scanner

Note Java naming convention
Method names – lowerCamelCase

next()
nextDouble()
nextInt()
nextLine()
…

hasNext()
hasNextDouble()
hasNextInt()
hasNextLine()
…



String	next(String pattern) Returns the next token if it matches the pattern constructed from the specified string.
BigDecimal	nextBigDecimal() Scans the next token of the input as a BigDecimal.
BigInteger	nextBigInteger() Scans the next token of the input as a BigInteger.
BigInteger	nextBigInteger(int radix) Scans the next token of the input as a BigInteger.
boolean	nextBoolean() Scans the next token of the input into a boolean value and returns that value.
byte	nextByte() Scans the next token of the input as a byte.
	nextByte(int radix) Scans the next token of the input as a byte.
	nextDouble() Scans the next token of the input as a double.
	nextFloat() Scans the next token of the input as a float.
	nextInt() Scans the next token of the input as an int.
	nextInt(int radix) Scans the next token of the input as an int.
	nextLine() Advances this scanner past the current line and returns the input that was skipped.

String Class: Representation in Text

- API documentation
  - http://docs.oracle.com/javase/8/docs/api/java/lang/String.html
- Import java.lang.String (optional)
- Ubiquitous; Has a rich set of methods

charAt()
concat()
equals()
indexOf()
lastIndexOf()
length()
toLowerCase()
toUpperCase()
substring()
trim()

And many more…

int	**indexOf**(int ch) Returns the index within this string of the first occurrence of the specified character.
int	**indexOf**(int ch, int fromIndex) Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
int	**indexOf**(String str) Returns the index within this string of the first occurrence of the specified substring.
int	**indexOf**(String str, int fromIndex) Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
String	**intern**() Returns a canonical representation for the string object.
boolean	**isEmpty**() Returns true if, and only if, **length()** is 0.
int	**lastIndexOf**(int ch) Returns the index within this string of the last occurrence of the specified character.
int	**lastIndexOf**(int ch, int fromIndex) Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
int	**lastIndexOf**(String str) Returns the index within this string of the last occurrence of the specified substring.
int	**lastIndexOf**(String str, int fromIndex) Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
int	**length**() Returns the length of this string.

String Class: Demo (1/2)

TestString.java

```
public class TestString {
   public static void main(String[] args) {
      String text = new String("I'm studying CS2040.");
      // or String text = "I'm studying CS2040.";

      System.out.println("text: " + text);
      System.out.println("text.length() = " + text.length());
      System.out.println("text.charAt(5) = " + text.charAt(5));
      System.out.println("text.substring(5,8) = " +
                         text.substring(5,8));
      System.out.println("text.indexOf(\"in\") = " +
                         text.indexOf("in"));

      String newText = text + "How about you?";
      newText = newText.toUpperCase();
      System.out.println("newText: " + newText);
      if (text.equals(newText))
         System.out.println("text and newText are equal.");
      else
         System.out.println("text and newText are not equal.");
   }
}
```

String Class: Demo (2/2)

Outputs

Explanations

```
text: I'm studying CS2040.
```

```
text.length() = 20
```

`length()` returns the length (number of characters) in **`text`**

```
text.charAt(5) = t
```

`charAt(5)` returns the character at position 5 in **`text`**

```
text.substring(5,8) = tud
```

`substring(5,8)` returns the substring in **`text`** from position 5 (‘**t**’) through position **7** (‘**d**’). ← ***Take note***

```
text.indexOf("in") = 9
```

`indexOf("in")` returns the starting position of "**in**" in **`text`**.

`newText = newText.toUppercase()`
converts characters in **`newText`** to uppercase.

```
newText: I'M STUDYING CS2040.HOW ABOUT YOU?
```

The + operator is string concatenation.

```
text and newText are not equal.
```

`equals()` compares two String objects.
Do **not** use `==`. (To be explained later.)

String Class: Comparing strings

- As strings are objects, do not use == if you want to check if two strings contain the same text
- Use the equals() method provided in the String class instead

```
Scanner sc = new Scanner(System.in);
System.out.println("Enter 2 identical strings:");
String str1 = sc.nextLine();
String str2 = sc.nextLine();

System.out.println(str1 == str2);
System.out.println(str1.equals(str2));
```

```
Enter 2 identical …
Hello world!
Hello world!
false
true
```

String Class: Immutable class

- String objects once created are immutable, that is you cannot change the content of the object
- This is why you see that all operations which “changes” the string actually returns a new string
- Not taking this into consideration can result in inefficient string processing
- For mutable strings you can look up StringBuilder in the Java API
- There are other immutable classes in Java API including all the primitive wrapper classes

Sequence and Subsequence

- A sequence is any enumerated collection of items in which repetition is allowed.
  - For example a sequence of integers <1,3,5,9,10>
  - A String can be considered a sequence of characters

- A subsequence is a possibly non-contiguous sequence of characters in a sequence. Relative ordering of items in subsequence is maintained

e.g **<snCS>** in **"I'm studying CS2040."**

e.g **<tud>** in **"I'm studying CS2040."**

e.g **<1,5,9>** in **<1,3,5,9,10>**

Math Class: Performing Computation

- API documentation
  - http://docs.oracle.com/javase/8/docs/api/java/lang/Math.html
- Import java.lang.Math (optional)

abs()
ceil()
floor()
hypot()
max()
min()
pow()
random()
sqrt()

And many more…

static double	**abs**(double a) Returns the absolute value of a double value.
static float	**abs**(float a) Returns the absolute value of a float value.
static int	**abs**(int a) Returns the absolute value of an int value.
static long	**abs**(long a) Returns the absolute value of a long value.
static double	**acos**(double a) Returns the arc cosine of a value; the returned angle is in the range 0.0 through *pi*.
static double	**asin**(double a) Returns the arc sine of a value; the returned angle is in the range -*pi*/2 through *pi*/2.
static double	**atan**(double a) Returns the arc tangent of a value; the returned angle is in the range -*pi*/2 through *pi*/2.
static double	**atan2**(double y, double x) Returns the angle *theta* from the conversion of rectangular coordinates (x, y) to polar coordinates (r, *theta*).
static double	**cbrt**(double a) Returns the cube root of a double value.
static double	**ceil**(double a) Returns the smallest (closest to negative infinity) double value that is greater than or equal to the argument and is equal to a mathematical integer.
static double	**copySign**(double magnitude, double sign) Returns the first floating-point argument with the sign of the second floating-point argument.
static float	**copySign**(float magnitude, float sign)

…nd floating-point argument.

static double	**E** The double value that is closer than any other to *e*, the base of the natural logarithms.
static double	**PI** The double value that is closer than any other to *pi*, the ratio of the circumference of a circle to its diameter.

2 class attributes (constants):
E and PI

Math Class: Demo

```
import java.util.*;

public class TestMath2 {
   public static void main(String[] ar
      Scanner sc = new Scanner(System.

      System.out.print("Enter 3 values
      double num1 = sc.nextDouble();
      double num2 = sc.nextDouble();
      double num3 = sc.nextDouble();

      System.out.printf("pow(%.2f,%.2f) = %.3f\n",
                        num1, num2, Math.pow(num1,num2));

      System.out.println("Largest = " +
                         Math.max(Math.max(num1,num2), num3));

      System.out.println("Generating 5 random values:");
      for (int i=0; i<5; i++)
         System.out.println(Math.random());
   }
}
```

```
Enter 3 values: 3.2 9.6 5.8
pow(3.20,9.60) = 70703.317
Largest = 9.6
Generating 5 random values:
0.87478272574496 5
0.948361014412348
0.896881621711305 3
0.028525690859603103
0.584650936426297 2
```

TestMath.java