CS1010 http://www.comp.nus.edu.sg/~cs1010/

Programming Methodology

UNIT 21

C to JAVA

CS1010 Final Examination (1/2)

- CS1010 Exam
  - 29 November 2017, Wednesday, 5 – 7 pm
  - Venue: To be announced by Registrar's Office
- Format
  - MCQs section: 6 questions (18 marks)
  - Short questions/Problem Solving section: 6 questions ranging from 5 to 26 marks (62 marks)
  - Total: 80 marks
- Grading of CS1010 is not based on bell curve

CS1010 Final Examination (2/2)

- Instructions
  - Open book
  - No calculators, electronic dictionaries and devices
  - May use 2B pencil to write programs
- Advice
  - Read instructions carefully and follow them!
  - Manage your time well!
  - A question may consist of several parts; the parts may be independent. If you are stuck on one part, move on to the next.

How to Prepare for Exams?

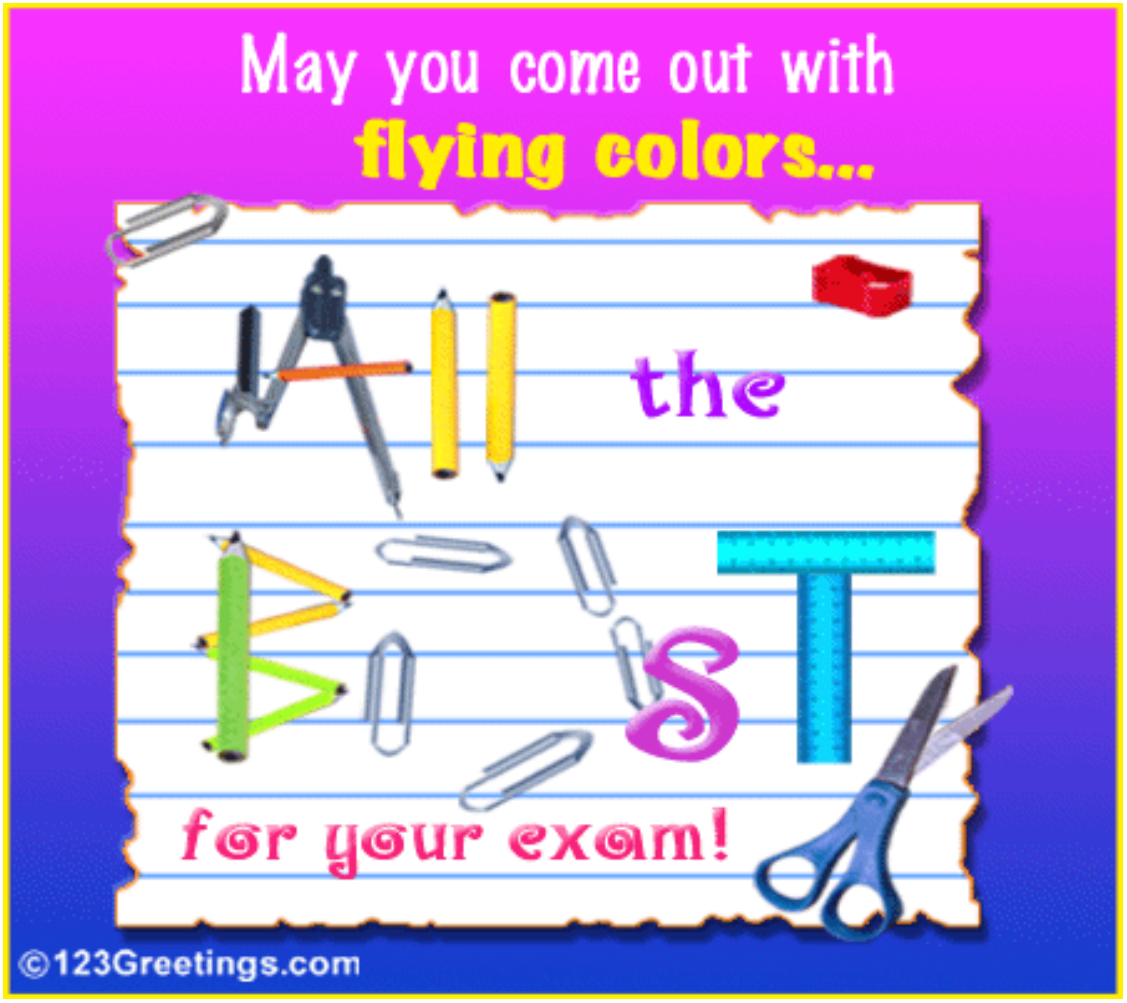
- Try past-years' exam papers
  - Answers not provided
  - Please discuss on IVLE forum
- Preparing for Exams: A Guide for NUS Students
  - http://www.cdtl.nus.edu.sg/Ufm/learn/36.htm

Post-CS1010

- For InfoSec/CEG: CS1010 → CS2040C
- For CS: CS1010 → CS2030 and CS2040
- CS2030 Programming Methodology II
  - Emphasis on object oriented and functional programming (software engineering)
  - Using Java, an object-oriented programming language
  - Website: http://www.comp.nus.edu.sg/~cs2030

Things-To-Do

- Keep an eye on the IVLE announcements on your CA marks
- Participate in IVLE forums
- Try out past years' exam papers
  - No solutions provided
  - Post on forums for clarification/discussion


May you come out with
flying colors...
the
for your exam!
©123Greetings.com

C to Java

Objectives:

- Learning about the structure of a Java program
- Using selection and repetition statements in Java
- Using classes such as Scanner, Math, String, DecimalFormat
- Learning how to use an array in Java
- With the above, able to convert some C programs written in CS1010 module to Java

C to Java (1/2)

1. Introduction
2. Hello World!
3. Java Constructs
   - 3.1 Program Structure
   - 3.2 Temperature Conversion
   - 3.3 Importing Packages
   - 3.4 Input: Using Scanner Class
   - 3.5 Output: Using System.out Class (with DecimalFormat class)
   - 3.6 Using Math Class
   - 3.7 User-defined Constants
   - 3.8 Java Naming Convention
   - 3.9 Writing Java Methods

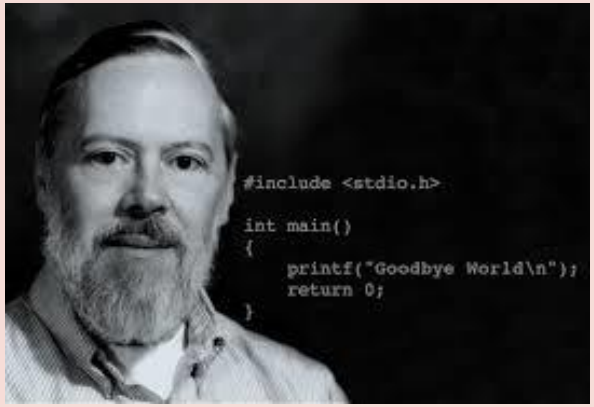
C to Java (2/2)

3. Java Constructs (continued…)
   - 3.10 Boolean Data Type
   - 3.11 Selection Statements
   - 3.12 Repetition Statements
   - 3.13 Arrays
4. API
5. Conclusion

1. Introduction (1/3)

- Java is the programming language used in CS2030.
- As an Object-Oriented Programming (OOP) language, Java supports OOP concepts such as encapsulation, inheritance and polymorphism. It also has exceptions handling (error handling) for better program robustness.
- In this unit, we will NOT discuss such OOP features. We will focus on non-OOP features in Java and compare the syntax of Java with that of C.
- However, even if we limit ourselves to non-OOP features here, certain OOP keywords such as ‘public’ and ‘static’ still creep up in our example Java programs. They require understanding of OOP concepts, which will be explained in CS2030. Here, we use them as a necessity.

1. Introduction (2/3)

	C	Java
When?	Created in 1972	Created in 1995
Who?	Dennis Ritchie 	James Gosling 
Where?	Bell Laboratories	Sun Microsystems (now merged with Oracle)

1. Introduction (3/3)

	C	Java
Type	Imperative	Imperative and object-oriented
Compilation	`gcc helloworld.c` → a.out (executable code)	`javac HelloWorld.java` → HelloWorld.class (bytecode)
Execution	`a.out`	`java HelloWorld`
Portability of compiled code	No, need to recompile for each architecture.	Yes, bytecode is “Write-Once, Run-Anywhere”.

2. Hello World! (1/2)

hello_world.c

```c
#include <stdio.h>

int main(void) {
   printf("Hello World!\n");
   return 0;
}
```

C programs: extension **.c**
Java programs: extension **.java**

C: Include header file <stdio.h> to use printf().
Java: Import java.lang.* package to use System.out.print()/println().

C: Must have main() function.
Java: Must have main() method.

HelloWorld.java

```java
import java.lang.*;

public class HelloWorld {

   public static void main(String[] args) {
      System.out.println("Hello World!");
   }
}
```

2. Hello World! (2/2)

C

```
happytan$ gcc hello_world.c
happytan$ a.out
Hello World!
```

Java

```
happytan$ javac HelloWorld.java
happytan$ java HelloWorld
Hello World!
```

3. Java

- Every Java program is a class.
- In OOP, a class is a template for creating objects (Analogy in C: structure definition in C is a template for creating structure variables)
- Such a class defines its object attributes and methods. We call it a service class. It does not contain a main() method. (Note: What we call functions in C, we call them **methods** in Java.)
- However, in this unit we are not writing service classes. We are writing classes that do not contain definition of any object attributes and methods. Instead, our classes are application (or client) classes that use other service classes. Such a class must contain the main() method.

3.1 Java: Program Structure

- For the moment just stick to the following program structure:

```
import java.lang.*;

public class ClassName {

   public static void main(String[] args) {
      // Body of main() method
   }
}
```

- java.lang is a package that contains the definition of methods such as System.out.print() and System.out.println().
- As java.lang is a default package, importing it is optional.
- The filename must be the same as the class name. Hence, the above program must be named **ClassName.java**
  - (Assuming the file contains one class, and the class is public.)

3.2 Java: Temperature Conversion

temperature_convert.c

```c
#include <stdio.h>
int main(void) {
   float fah, cel; // degrees Fahrenheit and Celsius

   printf("Enter temperature in Fahrenheit: ");
   scanf("%f", &fah);

   cel = (fah - 32) * 5/9;
   printf("Temperature in Celsius = %f\n", cel);

   return 0;
}
```

```
Enter ...: 123.5
... Celsius = 50.833332
```

TemperatureConvert.java

```java
import java.util.*;

public class TemperatureConvert {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      float fah, cel; // degrees Fahrenheit and Celsius

      System.out.print("Enter temperature in Fahrenheit: ");
      fah = sc.nextFloat();

      cel = (fah - 32) * 5/9;
      System.out.println("Temperature in Celsius = " + cel);
   }
}
```

3.3 Importing Packages

- Java comes with many built-in service classes with their methods, available for programmers to use.
- To use the methods from a service class, you need to import the respective package that contains that class.
- Examples:
  - To use System.out class → `import java.lang.*;` (default)
  - To use Scanner class → `import java.util.*;`
  - '*' is wildcard character, indicating all classes in that package; you may specifically import just that particular class of the package → `import java.util.Scanner;`

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

public class TemperatureConvert {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      ...
```

3.4 Input: Using Scanner Class (1/2)

- Scanner class (belongs to package java.util) provides methods to read inputs
- For interactive input, first, create a Scanner object to read data from System.in (keyboard)

```
Scanner sc = new Scanner(System.in);
```

- Then, use the appropriate Scanner methods to read the input data. Some examples:
  - `nextInt()`: To return integer read
  - `nextFloat()`: To return float value read
  - `nextDouble()`: To return double value read
  - `next()`: To return a string (delimited by whitespace) read
  - `nextLine()`: To return a string (delimited by newline) read

3.4 Input: Using Scanner Class (2/2)

int, double are **primitive data types** (just like in C)

String is a class in Java, in java.lang package.

InputExamples.java

```
Scanner sc = new Scanner(System.in);

int a = sc.nextInt();
double b = sc.nextDouble();
String str1 = sc.next();
String str2 = sc.nextLine();
```

User's inputs:

```
123
4.56
This is a test.
```

Values read:

a ← 123
b ← 4.56
str1 ←"This"
str2 ← " is a test."

3.5 Output: Using System.out Class (1/4)

- System.out class (belongs to package java.lang) provides methods to print outputs
- 3 methods:

```
System.out.print()
System.out.println()
System.out.printf()
```

- System.out.printf() is a relatively new method aiming to emulate the printf() function in C.

3.5 Output: Using System.out Class (2/4)

OutputExamples1.java

```
int a = 123, b = 456;
double x = 78.9;
String str = "Hello world!";

System.out.print("a = " + a);
System.out.println("; b = " + b);
System.out.print("x = " + x + " and ");
System.out.println("str = " + str);

System.out.printf("a = %d; b = %d\n", a, b);
System.out.printf("x = %f and ", x);
System.out.printf("str = %s\n", str);
```

Output:

```
a = 123; b = 456
x = 78.9 and str = Hello world!
a = 123; b = 456
x = 78.900000 and str = Hello world!
```

Note the difference in the display of x's value.

3.5 Output: Using System.out Class (3/4)

- What if you want to print a real number with a specific number of decimal places?
- Example: 2.37609 to display as 2.38?
- One way: using System.out.printf()

```
double y = 2.37609;
System.out.printf("y = %.2f\n", y);
```

- How about System.out.print() or System.out.println()?
- Use the DecimalFormat class (in java.text package)

3.5 Output: Using System.out with DecimalFormat (4/4)

OutputExamples2.java

```
DecimalFormat df1 = new DecimalFormat("000.00");
DecimalFormat df2 = new DecimalFormat("###.##");
DecimalFormat df3 = new DecimalFormat("0.0%");
DecimalFormat df4 = new DecimalFormat("0,000");
DecimalFormat df5 = new DecimalFormat("#,###");

double y = 2.37609, z = 36.9;
int a = 1234567, b = 89;

System.out.println("y = " + df1.format(y) + "; z = " + df1.format(z));
System.out.println("y = " + df2.format(y) + "; z = " + df2.format(z));
System.out.println("y = " + df3.format(y) + "; z = " + df3.format(z));
System.out.println("a = " + df4.format(a) + "; b = " + df4.format(b));
System.out.println("a = " + df5.format(a) + "; b = " + df5.format(b));
```

Many other format patterns available.

Output:

```
y = 002.38; z = 036.90
y = 2.38; z = 36.9
y = 237.6%; z = 3690.0%
a = 1,234,567; b = 0,089
a = 1,234,567; b = 89
```

3.6 Using Math Class (1/3)

- You have written C programs that use math functions in <math.h>
- Java provides the service class Math (in java.lang package)
- The Math class provides many math methods, and also some constants such as PI.
- Unlike in C where you need to compile a program that uses Math functions with the –lm option, no special option is required when you compile a Java program that uses Math methods.

3.6 Using Math Class (2/3)

area_of_circle.c

```
#include <stdio.h>
#include <math.h>
#define PI 3.14159

int main(void) {
  double radius, area;

  printf("Enter radius: ");
  scanf("%lf", &radius);
  area = PI * pow(radius, 2);
  printf("Area = %.3f\n", area);

  return 0;
}
```

```
Enter radius: 20.5
Area = 1320.253
```

3.6 Using Math Class (3/3)

AreaOfCircle.java

```
import java.util.*;
import java.text.*;

public class AreaOfCircle {

  public static void main(String[] args) {
    DecimalFormat df = new DecimalFormat("#.###");
    Scanner sc = new Scanner(System.in);
    double radius, area;

    System.out.print("Enter radius: ");
    radius = sc.nextDouble();
    area = Math.PI * Math.pow(radius, 2);

    System.out.println("Area = " + df.format(area));
  }
}
```

```
Enter radius: 20.5
Area = 1320.254
```

Constant PI defined in Math class.

Method pow() in Math class.

Area is slightly different from that in area_of_circle.c due to different values of PI.

3.7 User-defined Constants (1/3)

currency.c

```
#include <stdio.h>
#define SGD_TO_MYR 3.0269

int main(void) {
  double sing_dollar, ringgit;

  printf("Enter currency: S$");
  scanf("%lf", &sing_dollar);

  ringgit = SGD_TO_MYR * sing_dollar;
  printf("That's %.2f ringgit.\n", ringgit);

  return 0;
}
```

3.7 User-defined Constants (2/3)

Currency.java

```
import java.util.*;
import java.text.*;

public class Currency {
  private static final double SGD_TO_MYR = 3.0269;

  public static void main(String[] args) {
    DecimalFormat df = new DecimalFormat("0.00");
    Scanner sc = new Scanner(System.in);
    double singDollar, ringgit;

    System.out.print("Enter currency: S$");
    singDollar = sc.nextDouble();

    ringgit = SGD_TO_MYR * singDollar;
    System.out.println("That's " + df.format(ringgit)
                       + " ringgit.");
  }
}
```

3.7 User-defined Constants (3/3)

```
private static final double SGD_TO_MYR = 3.0269;
```

- The keyword final indicates that the value assigned is final, that is, it cannot be altered.
- The keyword private indicates that constant SGD_TO_MYR can only be used in this program.
- If you want to allow SGD_TO_MYR to be used by other programs, declare it as public instead. (Just like the PI constant in the Math class which is a public constant)
  - Other programs can then use this constant by referring to it as **Currency.SGD_TO_MYR**, just like **Math.PI**
- The keyword static is not within the scope of this unit.

3.8 Java Naming Convention (1/2)

- Java has a rather comprehensive naming convention on naming of classes, objects, methods, constants, etc.
  - http://geosoft.no/development/javastyle.html
- Some definitions: upper camel case, lower camel case
- Upper camel case:
  - First letter of each word is capitalised, the rest are in lower case
  - Examples: ThisIsAnExample, AreaOfCircle, TemperatureConvert
- Lower camel case:
  - Same as upper camel case, except that the first letter of the first word is in lower case
  - Examples: thisIsAnExample, singDollar, maxValue

3.8 Java Naming Convention (2/2)

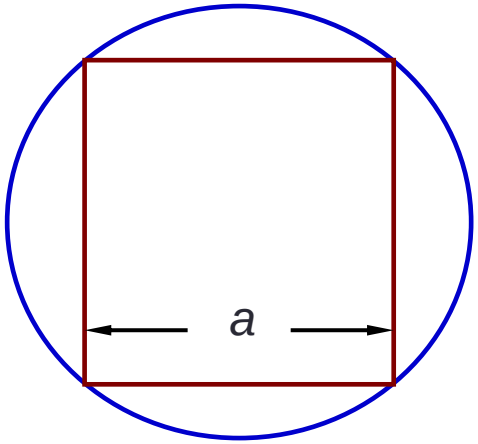
- Names of classes should be in upper camel case
  - Examples: AreaOfCircle, TemperatureConvert, Currency
- Names of variables, methods, parameters and objects should be in lower camel case:
  - Examples: singDollar, studentName, printArray() , bankAcct
- Names of constants should be in all upper case, with underscore (_) separating words:
  - Examples: PI, SGD_TO_MYR

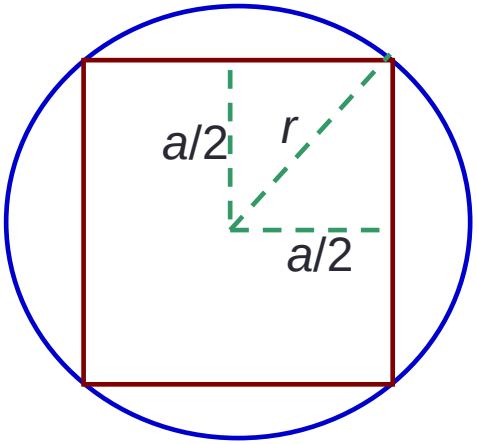
3.9 Writing Java Methods (1/6)

- What we call functions in C are called methods in Java.
- There are two kinds of methods in Java: instance methods and class methods.
- Instance methods: An instance method requires an object to apply (pass message) to.
- Class methods: A class method does not require an object to apply (pass message) to.
- As we are not doing OOP here, we will write only class methods here.
- Class methods are distinguished from instance methods by the presence of the keyword static.

3.9 Writing Java Methods (2/6)

- If *a* is the length of the square, what is the area of the circle?





- Let radius of circle be *r*
- Pythagoras' theorem

$$r^2 = (a/2)^2 + (a/2)^2$$

$$= a^2/2$$

- Hence, area of circle, *C*, is

$$C = \pi \times a^2/2$$

3.9 Writing Java Methods (3/6)

square_and_circle.c

```c
#include <stdio.h>
#define PI 3.14159

double compute_area(double);

int main(void) {
  double length; // length of the square
  double area;   // area of the circle

  printf("Enter length of square: ");
  scanf("%lf", &length);
  area = compute_area(length);
  printf("Area of circle = %.3f\n", area);

  return 0;
}

double compute_area(double length) {
  return PI * (length * length)/2;
}
```

3.9 Writing Java Methods (4/6)

SquareAndCircle.java

```
import java.util.*;
import java.text.*;

public class SquareAndCircle {

  public static void main(String[] args) {
    DecimalFormat df = new DecimalFormat("#.###");
    Scanner sc = new Scanner(System.in);
    double length; // length of the square
    double area;   // area of the circle

    System.out.print("Enter length of square");
    length = sc.nextDouble();
    area = computeArea(length);
    System.out.println("Area of circle "
                       + df.format(area));
  }
  // continue on next page
```

3.9 Writing Java Methods (5/6)

SquareAndCircle.java

```
public class SquareAndCircle {

  public static void main(String[] args) {
    . . .
    area = computeArea(length);
    . . .
  }

  public static double computeArea(double length) {
    return Math.PI * (length * length)/2;
  }
}
```

- Almost like function in C; function prototype not needed
- Add keywords public (change to private if you don't want to share this method) and static (not in scope here)

3.9 Writing Java Methods (6/6)

- In C, for a function to pass back more than one value to the caller, we use pointer parameters.
- There are no pointers in Java.
- In Java, we create an object to hold the values, and pass the reference of the object to the method.
- This is outside the scope of this unit.

3.10 Boolean Data Type (1/2)

- Java provides the boolean data type, which is absent in ANSI C

BooleanType.java

```
public class BooleanType {

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a: ");
    int a = sc.nextInt();
    boolean b = (a > 0);
    System.out.println("(a > 0) is " + b);
  }
}
```

```
Enter a: 20
(a > 0) is true
```

```
Enter a: -3
(a > 0) is false
```

3.10 Boolean Data Type (2/2)

- In ANSI C, zero represents false and any other value represents true.

C

```
if (a%2 == 0)
  printf("a is even\n");
else
  printf("a is odd\n");
```

```
if (a%2)
  printf("a is odd\n");
else
  printf("a is even\n");
```

- This is not the case for Java.

Java

```
if (a%2 == 0) System.out.println("a is even");
else          System.out.println("a is odd");
```

```
if (a%2) System.out.println("a is odd");
else     System.out.println("a is even");
```

Compilation error!

3.11 Selection Statements (1/2)

- Selection statements in Java – 'if-else' and 'switch' – are the same as those in C

```
if ((age < 1) || (age > 120))
  System.out.println("Invalid age.");
else if (age >= 18)
  System.out.println("Eligible for driving licence.");
```

```
System.out.println("Enter marks (0-100): ");
int marks = sc.nextInt();
char grade;
switch (marks/10) {
  case 8: case 9: case 10: grade = 'A'; break;
  case 7: grade = 'B'; break;
  case 6: grade = 'C'; break;
  case 5: grade = 'D'; break;
  default: grade = 'F';
}
```

Grade.java

3.11 Selection Statements (2/2)

- Short-circuit evaluation applies in Java as in C

```
if ((count != 0) && (sum/count > 0.5)) {
   ...
}
```

- If count is zero, the second boolean expression (sum/count > 0.5) will not be evaluated.

3.12 Repetition Statements

- Repetition statements in Java – 'while', 'do-while' and 'for' – are also the same as those in C

```
int i = 1, sum = 0;
while (sum < 1000) {
  sum += i;
  i++;
}
```

```
do {
  age = sc.nextInt();
} while ((age<1) || (age>120));
```

```
for ( int i=0; i<10; i++ ) {
  System.out.println(i);
}

System.out.println(i);
```

Java allows loop variable to be declared in the 'for' block; however, the scope of variable **i** is bound within the block.

Compilation error as variable **i** is not recognised outside the 'for' block.

3.13 Arrays (1/4)

- In Java, an array is an object, and has a public attribute length which is the number of elements of the array.
- Declaration:

C
```
int arr[8];
```

Java
```
int[] arr = new int[8];
```

- Initialisation:

C
```
double arr[] = { 1.2, 3.5, 2.7 };
```

Java
```
double[] arr = { 1.2, 3.5, 2.7 };
```

3.13 Arrays (2/4)

ArrayDemo1.java

```
public class ArrayDemo1 {

  public static void main(String[] args) {
    int[] arr;

    // create a new integer array with 3 elements
    // arr now refers (points) to this new array
    arr = new int[3];

    // using the 'length' attribute
    System.out.println("Length = " + arr.length);

    arr[0] = 100;
    arr[1] = arr[0] - 37;
    arr[2] = arr[1] / 2;
    for (int i=0; i<arr.length; i++) {
      System.out.println("arr[" + i + "] = " + arr[i]);
    }
  }
}
```

```
Length = 3
arr[0] = 100
arr[1] = 63
arr[2] = 31
```

3.13 Arrays: Passing Array to a Method (3/4)

ArrayDemo2.java

```
public class ArrayDemo2 {

  public static void main(String[] args) {
    int[] arr = new int[3];
    System.out.println("Length = " + arr.length);
    arr[0] = 100;
    arr[1] = arr[0] - 37;
    arr[2] = arr[1] / 2;
    printArray(arr);
  }

  public static void printArray(int[] a) {
    for (int i=0; i<a.length; i++) {
      System.out.println("a[" + i + "] = " + a[i]);
    }
  }
}
```

```
Length = 3
a[0] = 100
a[1] = 63
a[2] = 31
```

3.13 Arrays: Passing Array to a Method (4/4)

ArrayDemo3.java

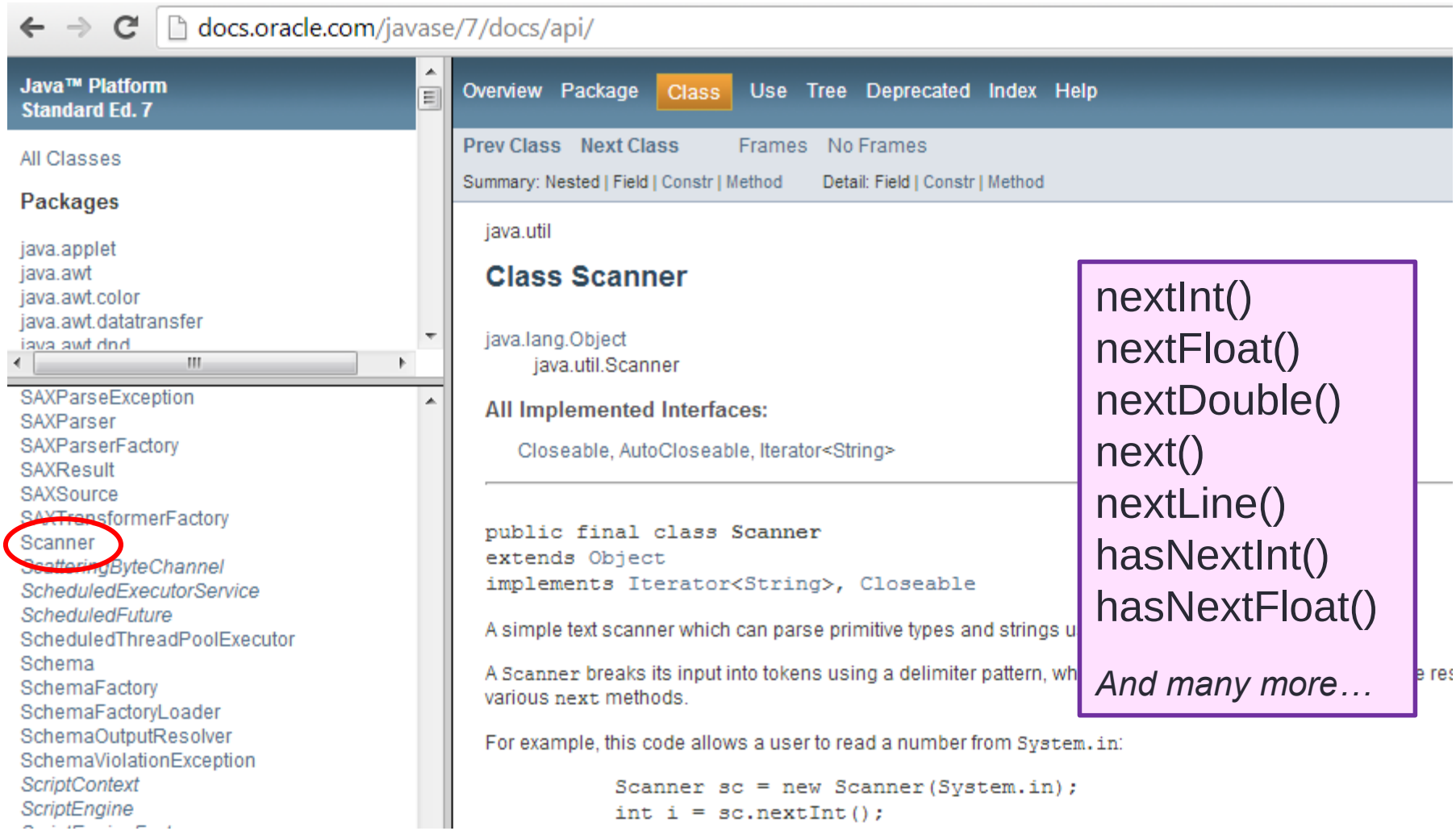
```
import java.util.*;
public class ArrayDemo3 {

  public static void main(String[] args) {
     int[] arr = new int[3];
     scanArray(arr);
     System.out.println("Sum = " + sumArray(arr));
  }

  public static void scanArray(int[] a) {
     Scanner sc = new Scanner(System.in);
     System.out.print("Enter " + a.length + " values: ");
     for (int i=0; i<a.length; i++)
       a[i] = sc.nextInt();
  }

  public static int sumArray(int[] a) {
     int sum = 0;
     for (int i=0; i<a.length; i++)
       sum += a[i];
     return sum;
  }
}
```

```
Enter 3 values: 16 32 8
Sum = 56
```

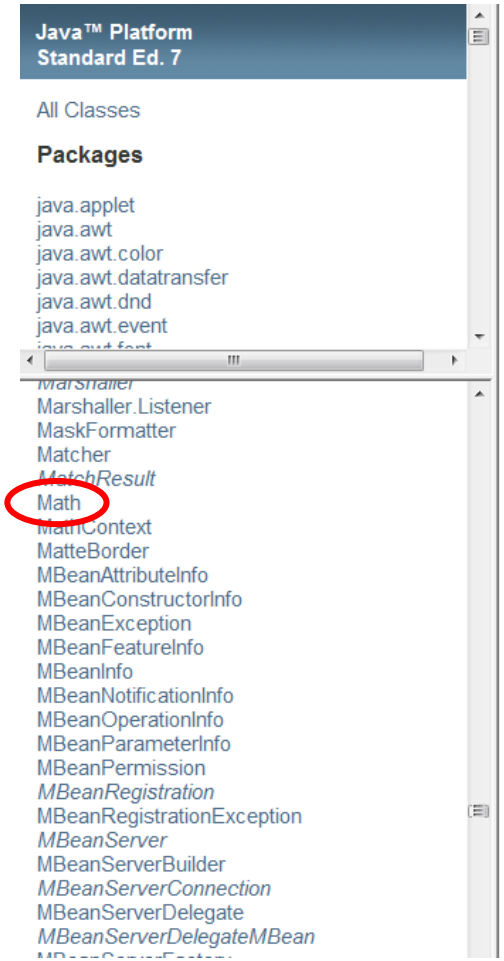
4. API (1/4)

- The service classes we have used (Scanner, String, Math, DecimalFormat) are all parts of the Java API (Application Programming Interface)
  - API: an interface for other programs to interact with a program without having direct access to the internal data of the program
  - Documentation, SE8: http://docs.oracle.com/javase/8/docs/api/
  - For Java programmers, it is very important to refer to the API documentation regularly!
- The API consists of many classes (hundreds of them!)

4. API: Scanner Class (2/4)



4. API: Math Class (3/4)



Modifier and Type	Field and Description
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 cir

Method Summary

Methods

Modifier and Type	Method and Description
static double	**abs**(double a) Returns the absolute value of a doubl
static float	**abs**(float a) Returns the absolute value of a float
static int	**abs**(int a) Returns the absolute value of an int
static long	**abs**(long a) Returns the absolute value of a long
static double	**acos**(double a) Returns the arc cosine of a value; the re
static double	**asin**(double a) Returns the arc sine of a value; the retu
static double	**atan**(double a) Returns the arc tangent of a value; the returned angle is in the range -*pi*/2 through *pi*/2.

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

And many more…

4. API: String Class (4/4)

- Ubiquitous
- Has a rich set of methods to manipulate strings

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.
boolean	**matches**(String regex)

5. Conclusion

- We have attempted to confine this unit to non-OOP aspects of Java
- The aim is to allow you to understand the basic Java (non-OOP) constructs and…
- To convert some of the C programs you have written in CS1010 to Java programs

Summary

- In this unit, you have learned about
  - The program structure of Java
  - The non-OOP aspects of Java programming
    - Variables, constants, methods, selection and repetition statements
    - Using classes (Scanner, String, Math, DecimalFormat) and packages
    - Java naming convention
  - Declaration and use of arrays
  - API: The Java documentation