

Object Oriented Programming

Object Oriented Programming (OOP) Part 1 – User Mode

A paradigm shift:

From procedural to object-oriented model

Acknowledgement

- The contents of these slides have origin from School of Computing, National University of Singapore.
- We greatly appreciate support from Mr. Aaron Tan Tuck Choy, and Dr. Low Kok Lim for kindly sharing these materials.

Policies for students

- These contents are only used for students PERSONALLY.
- Students are NOT allowed to modify or deliver these contents to anywhere or anyone for any purpose.

Objectives

Java

- (Re)introducing API
- Using Java classes
- Basic features/concepts of OOP

References



Textbook

- Chapter 2: Section 2.2 (pg 119 – 130), Section 2.3 (pg 131 – 150)
- String class: Section 1.5 (pg 59 – 64)
- Wrapper classes: Section 1.1 (pg 29 – 30)

Outline (1/2)

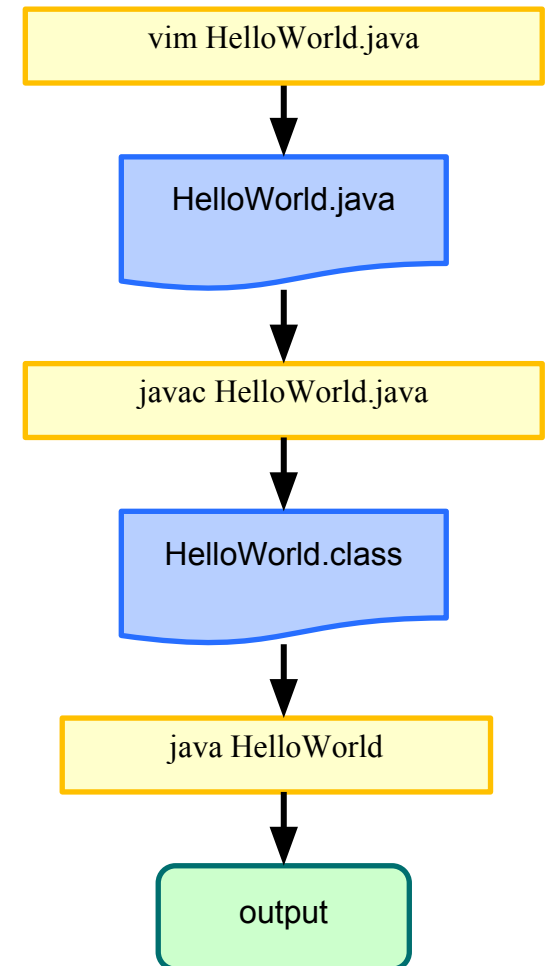
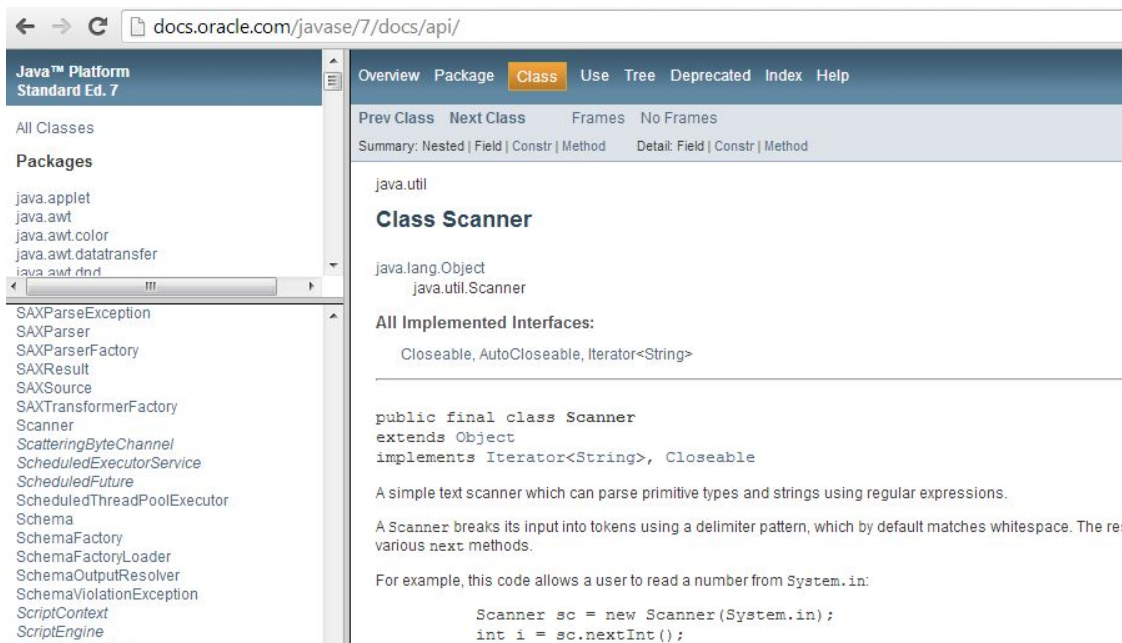
1. Recapitulation
2. API: Where you find service classes
 - 2.1 Scanner class (revisit)
 - 2.2 String class (revisit)
 - 2.3 Math class (revisit)
3. OOP concepts (basic)
 - 3.1 Modifiers
 - 3.2 Class vs Instance methods
 - 3.3 Constructors
 - 3.4 Overloading

Outline (2/2)

- 4. More classes (new)
 - 4.1 DecimalFormat class
 - 4.2 Random class
 - 4.3 Wrapper classes
 - 4.4 Point class
- 5. Abstraction and Information Hiding
 - 5.1 What is Abstraction?
 - 5.2 Procedural Abstraction

1. Recapitulation

- Compiling and running Java programs
- Java program structure
- Basic Java elements
- API: Scanner class, Math class



2. API (Revisit)

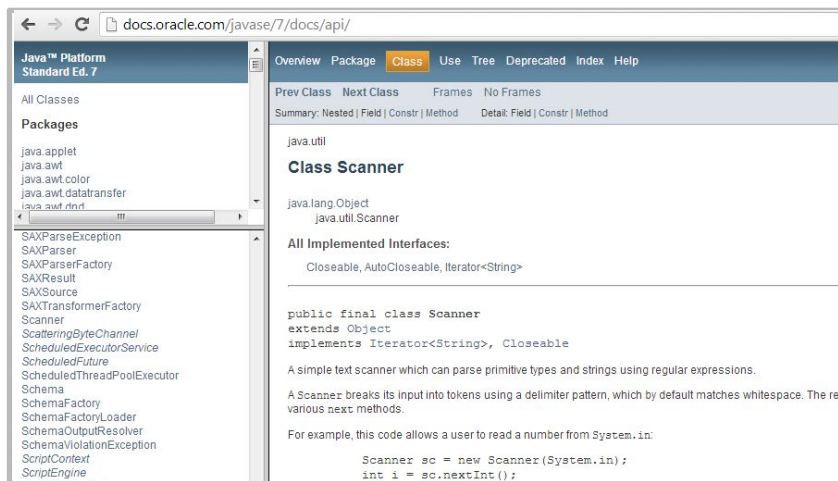


Application Programming Interface –
Where you find service classes

Java Programmer

API Specification

<https://docs.oracle.com/en/java/javase/11/docs/api/>



Last week:

Scanner class 

String class 

Math class 

*And from now on,
many many more...*

Scanner Class: Reading Inputs

- API documentation
 - <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/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()`
...

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

Scanner Class: Demo (1/2)

```
Enter name1: Wilson Wee
name1 entered is ???
Enter name2: Wilson Wee
name2 entered is ???
```

```
import java.util.*;

public class TestScanner {

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

        // Comparing nextLine() and next()
        System.out.print("Enter name1: ");
        String name1 = sc.nextLine();
        System.out.println("name1 entered is '" + name1 + "'.");

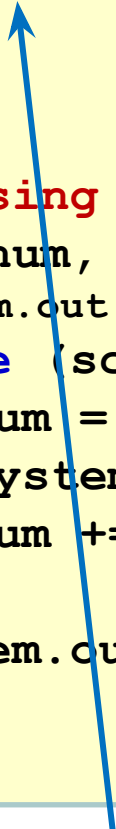
        System.out.print("Enter name2: ");
        String name2 = sc.next();
        System.out.println("name2 entered is '" + name2 + "'.");
    }
}
```

TestScanner.java

Scanner Class: Demo (2/2)

```
sc.nextLine(); // to skip the rest of the line after
               // the next() method captured the
               // first word of the second name

// Using nextInt() and hasNextInt()
int num, sum = 0;
System.out.println("Enter integers, terminate with control-d:");
while (sc.hasNextInt()) {
    num = sc.nextInt();
    System.out.println("Integer read: " + num);
    sum += num;
}
System.out.println("Sum = " + sum)
}
```



```
Enter integers, ...
17
Integer read: 17
5
Integer read: 5
(More will be shown in
lecture)
```

What is this for?
Attend lecture for explanation!

Scanner Class: For CodeCrunch



- For a program to work in CodeCrunch, it **must not have more than one Scanner object.**
- Hence, create at most one Scanner object and use it to read all inputs.

String Class: Representation in Text

- API documentation
 - <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/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.
boolean	matches (String regex)	

String Class: Demo (1/2)

TestString.java

```
public class TestString {  
    public static void main(String[] args) {  
        String text = new String("I'm studying 503005.");  
        // or String text = "I'm studying 503005.";  
        // We will explain the difference later.  
        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.");  
    }  
}
```


2. API String Class: Demo (2/2)

Outputs

```
text: I'm studying 503005.
```

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

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

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

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

```
newText = newText.toUpperCase()  
converts characters in newText to uppercase.
```

```
newText: I'M STUDYING 503005.HOW ABOUT YOU?
```

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

Explanations

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

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

substring(5,8) returns the substring in **text** from position 5 ('t') through position 7 ('d'). [Take note](#)

indexOf("in") returns the ...?

The **+** operator is string concatenation.

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 (more details about **equals()** in next lecture)

```
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!  
(What will be printed?)
```

Math Class: Performing Computation

- API documentation
 - <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Math.html>
- Import `java.lang.Math` (optional)

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

And many more...

2 class attributes
 (constants):
E and **PI**

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 π .
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 <i>theta</i> from the conversion of rectangular coordinates (x, y) to polar coordinates (r, <i>theta</i>).
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)	

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 π , the ratio of the circumference of a circle to its diameter.

Math Class: Demo

- A demo was given last week. Here's another

```
import java.util.*;

public class TestMath2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        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:
0.874782725744965
0.948361014412348
0.8968816217113053
0.028525690859603103
0.5846509364262972
```

TestMath2.java

3. OOP Concepts

What makes Java object-oriented?

Modifiers

: keywords added to specify the way a class/attribute/method works

```
public class TestMath2
```

```
public static void main(String[] args)
```

Access-control
modifiers: public,
private, etc.

Non-access
modifiers

More about 'public' and
'private' in next lecture.

```
String
```

```
trim()
```

Returns a copy of the string, with
whitespace omitted.

```
static String
```

```
valueOf(boolean b)
```

Returns the string representatio
argument.

```
static String
```

```
valueOf(char c)
```

Returns the string representatio
argument.

```
static String
```

```
valueOf(char[] data)
```

Class vs Instance methods (1/4)

Math

String class

String	trim()	Returns a copy of the str whitespace omitted.
static String	valueOf(boolean b)	Returns the string repres argument.
static String	valueOf(char c)	Returns the string repres argument.

static float	signum(float f)	Returns the signum function of the zero, 1.0f if the argument is greater less than zero.
static double	sin(double a)	Returns the trigonometric sine of a
static double	sinh(double x)	Returns the hyperbolic sine of a do
static double	sqrt(double a)	Returns the correctly rounded posit value.
static double	tan(double a)	Returns the trigonometric tangent c

- A **static method** (preferably called a **class method**) means that no object (instance) of the class is needed to use the method.
- A **non-static method** (preferably called an **instance method**) means that the method must be applied to **an object (instance)** of that class.

Scanner class

float	nextFloat()	Scans the next token of the inp
int	nextInt()	Scans the next token of the inp
int	nextInt(int radix)	Scans the next token of the inp
String	nextLine()	Advances this scanner past th was skipped.

Class vs Instance methods (2/4)

Math

String class

String	trim()	Returns a copy of the string with all whitespace omitted.
static String	valueOf(boolean b)	Returns the string representation of the boolean argument.
static String	valueOf(char c)	Returns the string representation of the char argument.

class

static float	signum(float f)	Returns the signum function of the argument. Returns 1.0f if the argument is greater than zero, -1.0f if the argument is less than zero, and 0.0f if the argument is zero.
static double	sin(double a)	Returns the trigonometric sine of the argument.
static double	sinh(double x)	Returns the hyperbolic sine of the argument.
static double	sqrt(double a)	Returns the correctly rounded positive square root of the argument.
static double	tan(double a)	Returns the trigonometric tangent of the argument.

■ Observations

- All methods in the **Math** class are class methods.
- All methods in the **Scanner** class are instance methods.
- The **String** class comprises a mix of class and instance methods.

Scanner class

float	nextFloat()	Scans the next token of the input as a float.
int	nextInt()	Scans the next token of the input as an int.
int	nextInt(int radix)	Scans the next token of the input as an int in the specified radix.
String	nextLine()	Advances this scanner past the current line. The method returns the text that was skipped, including the line separator.

Class vs Instance methods (3/4)

- Calling a **class method**

```
double answer = Math.pow(3.5, 2.2);
```

Precede method with the class name

```
public class Exercise {  
    public static double volumeCone(double rad, double ht) {  
        return Math.PI * rad * rad * ht / 3.0;  
    }  
}
```

```
public static void main(String[] args)  
    . . .
```

Optional to precede method with the class name if the method is defined in the class it is called.

```
double vol = volumeCone(radius, height);  
/* Alternatively:  
double vol = Exercise.volumeCone(radius, height);  
*/  
}
```

Class vs Instance methods (4/4)

- Calling an **instance method**

```
int value = Scanner.nextInt();
```

WRONG!

```
// create an instance (object) of Scanner  
Scanner sc = new Scanner(System.in);  
int value = sc.nextInt();
```

RIGHT!

```
String str = "Some text";  
str = String.toUpperCase();
```

WRONG!

```
String str = "Some text";  
str = str.toUpperCase();
```

RIGHT!

- ❑ An instance method must be applied to an instance (object) of a class.
- ❑ “**Calling an instance method**” is sometimes referred to as “**passing a message to an instance (object)**”.

Class methods in String class

- We have used instance methods in **String** class, but not class methods
- Some class methods in **String** class:

<code>static String</code>	<code>valueOf(double d)</code> Returns the string representation of the <code>double</code> argument.
<code>static String</code>	<code>valueOf(float f)</code> Returns the string representation of the <code>float</code> argument.
<code>static String</code>	<code>valueOf(int i)</code> Returns the string representation of the <code>int</code> argument.

```
String str = String.valueOf(123);
```

What does `str` contain after the above statement?

Constructors (1/2)

- When a class (eg: String, Scanner) provides instance methods, it expects instances (objects) to be created from that class
- This requires a special method called a **constructor**

Constructor and Description

Scanner(File source)

Constructs a new `Scanner` that produces values scanned from the specified file.

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

Note the keyword **new**

Constructor and Description

String()

Initializes a newly created `String` object so that it represents an empty character sequence.

String(String original)

Initializes a newly created `String` object so that it represents the same sequence of characters

```
String str1 = new String();  
String str2 = new String("To be or not to be?");
```

Constructors (2/2)

- The keyword **new** is used to invoke the constructor
- Exception: **String** class

```
String str1 = new String();  
String str2 = new String("To be or not to be?");
```



*Somewhat equivalent **

```
String str1 = "";  
String str2 = "To be or not to be?";
```

* Just for today's purpose. The 2 ways of constructing a string are not exactly equivalent though.

- **String** is a special class
 - Has an alternative syntax to construct a String object
 - String objects are **immutable**
 - More about Strings (to be explored in tutorial)

Overloading

- Observe that some methods have identical names, but with different parameters. This is called **overloading**.

Math class

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.

Overloaded methods

- Without overloading, different named methods would have to be provided:
 - absDouble**(double a)
 - absFloat**(float a)
 - absInt**(int a)
 - absLong**(long a)
- With overloading, all these related methods have the same name.

String class

String	substring (int beginIndex)	Returns a new string that is a substring of this string.
String	substring (int beginIndex, int endIndex)	Returns a new string that is a substring of this string.

Constructor and Description

String ()	Initializes a newly created string object so that it represents an empty character sequence.
String (String original)	Initializes a newly created string object so that it represents the same sequence of characters as the specified String object.

Overloaded constructors

4. More Classes

Many classes in Java API!

DecimalFormat Class (1/3)

- We have used the `System.out.printf()` statement to format the output of real number

```
System.out.printf("Math.PI = %.3f\n", Math.PI);
```

```
Math.PI = 3.142
```

- Alternatively, you may use the `DecimalFormat` class
 - Import `java.text` package

DecimalFormat Class (2/3)

Symbol	Location	Localized?	Meaning
0	Number	Yes	Digit
#	Number	Yes	Digit, zero shows as absent
.	Number	Yes	Decimal separator or monetary decimal separator
-	Number	Yes	Minus sign
,	Number	Yes	Grouping separator
E	Number	Yes	Separates mantissa and exponent in scientific notation. <i>Need not be quoted in prefix or suffix.</i>
;	Subpattern boundary	Yes	Separates positive and negative subpatterns
%	Prefix or suffix	Yes	Multiply by 100 and show as percentage

Example:

```
DecimalFormat df = new DecimalFormat("0.000");
```

¤ (\u00A4)	Prefix or suffix	No	Currency sign, replaced by currency symbol. If doubled, replaced by international currency symbol. If present in a pattern, the monetary decimal separator is used instead of the decimal separator.
'	Prefix or suffix	No	Used to quote special characters in a prefix or suffix, for example, "'#'" formats 123 to "#123". To create a single quote itself, use two in a row: "##'clock".

<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/text/DecimalFormat.html>

DecimalFormat Class (3/3): Example

TestDecimalFormat.java

```
import java.text.DecimalFormat;

public class TestDecimalFormat {
    public static void main(String[] args) {
        DecimalFormat df1 = new DecimalFormat("0.000"); // 3 dec.
        pl.

        DecimalFormat df2 = new DecimalFormat("#.###");
        DecimalFormat df3 = new DecimalFormat("0.00%");

        System.out.println("PI = " + df1.format(Math.PI));
        System.out.println("12.3 formatted with \"0.000\" = "
            + df1.format(12.3));
        System.out.println("12.3 formatted with \"#.###\" = "
            + df2.format(12.3));
        System.out.println("12.3 formatted with \"0.00%\" = "

    }
}
```

```
PI = 3.142
12.3 formatted with "0.000" = 12.300
12.3 formatted with "#.###" = 12.3
12.3 formatted with "0.00%" = 1230.00%
```

- Note that **df.format(x)** does not change the value **x**. It merely displays the value **x** in the specified format.

Random Class (1/4)

- Sometimes we may need to generate random numbers for some applications, such as simulation or to fill an array with random values
- The **Math** class provides a **random()** method

<code>static double</code>	<code>random()</code> Returns a <code>double</code> value with a positive sign, greater than or equal to 0.0 and less than 1.0.
----------------------------	---

- Alternatively, you may use the **Random** class
 - ❑ Import **java.util** package
 - ❑ <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Random.html>

Random Class (2/4)

- Constructors
 - ❑ `Random()`: random numbers generated are different each time program is run
 - ❑ `Random(long seed)`: random numbers generated are taken from a pre-determined fixed sequence based on the seed

Constructors

Constructor and Description

`Random()`

Creates a new random number generator.

`Random(long seed)`

Creates a new random number generator using a single `long` seed.

Random Class (3/4)

- Some methods in **Random** class

<code>double</code>	<code>nextDouble()</code> Returns the next pseudorandom, uniformly distributed <code>double</code> value between 0.0 and 1.0 from this random number generator's sequence.
<code>float</code>	<code>nextFloat()</code> Returns the next pseudorandom, uniformly distributed <code>float</code> value between 0.0 and 1.0 from this random number generator's sequence.
<code>double</code>	<code>nextGaussian()</code> Returns the next pseudorandom, Gaussian ("normally") distributed <code>double</code> value with mean 0.0 and standard deviation 1.0 from this random number generator's sequence.
<code>int</code>	<code>nextInt()</code> Returns the next pseudorandom, uniformly distributed <code>int</code> value from this random number generator's sequence.
<code>int</code>	<code>nextInt(int n)</code> Returns a pseudorandom, uniformly distributed <code>int</code> value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.

Random Class (4/4): Example

TestRandom.java

```
import java.util.Random;

public class TestRandom {
    public static void main(String[] args) {
        // To generate a random integer in [51,70]
        // using Math.random() and Random's
        nextInt()
        int num1 = (int) (Math.random() * 20) + 51;

        System.out.println("num1 = " + num1);

        Random rnd = new Random();
        int num2 = rnd.nextInt(20) + 51;

        System.out.println("num2 = " + num2);
    }
}
```

```
num1 = 51
num2 = 68
```

int

nextInt(int n)

Returns a pseudorandom, uniformly distributed `int` value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.

Wrapper Classes (1/2)

- Object-oriented counterparts of primitive data types
- Types such as `int`, `float`, `double`, `char`, `boolean`, etc. are **primitive data types**.
 - They are not objects. They are legacies of older languages.
- Sometimes we need object equivalent of these primitive data types (when we cover more advanced OOP concepts later)
- These are called **wrapper classes** – one wrapper class corresponding to each primitive data type

Primitive data type	Wrapper class
<code>int</code>	<code>Integer</code>
<code>long</code>	<code>Long</code>
<code>float</code>	<code>Float</code>
<code>double</code>	<code>Double</code>
<code>char</code>	<code>Character</code>
<code>boolean</code>	<code>Boolean</code>
<i>and others...</i>	

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 API documentation and explore the wrapper classes on your own



# Point Class (1/5)

- An OOP program allows the creation of **instances** (also called **objects**) of a **class** and passing **messages** to these objects (calling methods on these objects)
- We have used **Scanner** and **String** classes
- We introduce another class, **Point**, which contains a number of OOP concepts we will explore in more depth in next lecture
  - Import **java.awt** package
  - <https://docs.oracle.com/en/java/javase/11/docs/api/java.desktop/java/awt/Point.html>



## Point Class (2/5): Attributes

- The **Point** class contains 2 **attributes**
  - Sometimes also called **data members**
  - In the API documentation, they are labelled as **fields**
- Attributes can be **class attributes** (with **static** modifier) or **instance attributes** (without **static** modifier)
  - Details to be covered in next lecture
- The 2 attributes in **Point** class are instance attributes: **x** and **y**, representing the x- and y-coordinates

| Fields            |                                                           |
|-------------------|-----------------------------------------------------------|
| Modifier and Type | Field and Description                                     |
| int               | <b>x</b><br>The X coordinate of this <code>Point</code> . |
| int               | <b>y</b><br>The Y coordinate of this <code>Point</code> . |

# Point Class (3/5): Constructors

- These are the overloaded constructors in **Point** class

## Constructors

### Constructor and Description

**Point()**

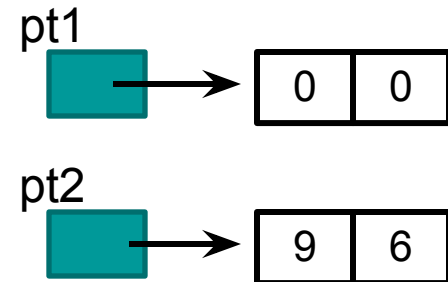
Constructs and initializes a point at the origin (0, 0) of the coordinate space.

**Point(int x, int y)**

Constructs and initializes a point at the specified (x, y) location in the coordinate space.

**Point(Point p)**

Constructs and initializes a point with the same location as the specified **Point** object.



- Examples:

```
Point pt1 = new Point(); // pt1 is (0, 0)
Point pt2 = new Point(9, 6); // pt2 is (9, 6)
```

# Point Class (4/5): Methods

## ■ Methods in **Point** class

| Methods           |                                                                                                                                                                                  |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Modifier and Type | Method and Description                                                                                                                                                           |
| boolean           | <b>equals</b> (Object obj)<br>Determines whether or not two points are equal.                                                                                                    |
| <b>Point</b>      | <b>getLocation</b> ()<br>Returns the location of this point.                                                                                                                     |
| double            | <b>getX</b> ()<br>Returns the X coordinate of this <code>Point2D</code> in double precision.                                                                                     |
| double            | <b>getY</b> ()<br>Returns the Y coordinate of this <code>Point2D</code> in double precision.                                                                                     |
| void              | <b>move</b> (int x, int y)<br>Moves this point to the specified location in the (x, y) coordinate plane.                                                                         |
| void              | <b>setLocation</b> (double x, double y)<br>Sets the location of this point to the specified double coordinates.                                                                  |
| void              | <b>setLocation</b> (int x, int y)<br>Changes the point to have the specified location.                                                                                           |
| void              | <b>setLocation</b> (Point p)<br>Sets the location of the point to the specified location.                                                                                        |
| <b>String</b>     | <b>toString</b> ()<br>Returns a string representation of this point and its location in the (x, y) coordinate space.                                                             |
| void              | <b>translate</b> (int dx, int dy)<br>Translates this point, at location (x, y), by dx along the x axis and dy along the y axis so that it now represents the point (x+dx, y+dy). |

# Point Class (5/5): Demo

TestPoint.java

```
// To test out Point class
import java.util.*;
import java.awt.*;

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

 System.out.print("Enter x and y: ");
 int xCoord = sc.nextInt();
 int yCoord = sc.nextInt();

 Point pt = new Point(xCoord, yCoord);

 System.out.println("x-coordinate is " + pt.getX());
 System.out.println("y-coordinate is " + pt.y);

 System.out.println("The point created is " + pt);
 // or: System.out.println("The ... is " +
 pt.toString());
 }
}
```

Enter x and y: 12 -7  
 x-coordinate is 12.0 ← Note: getX() returns double  
 y-coordinate is -7  
 The point created is java.awt.Point[x=12,y=-7]

To be discussed in next lecture.

# Common Mistake

- Accessing an object before it is created

```
Point pt;
pt.setLocation(12,10); // change coordinates of pt
```

**WRONG!**

pt

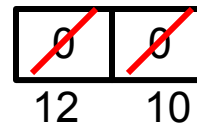


The Point object  
does not even exist!

```
Point pt = new Point(); // create Point object pt
pt.setLocation(12,10); // change coordinates of pt
```

**RIGHT!**

pt



# FAQ



- Q: Must we know all the classes on the API?
- A: There are hundreds of them, so you cannot possibly know all of them. 😊 You are expected to know those covered in lectures, labs, tutorials and any additional materials given out.
- **Familiarity** is the key, so you need to **practice a lot**, and **refer to the API document as often as possible**. There are many things not covered in class but you can explore on your own.
- Like 501042 (or equivalent), you must be prepared to invest time in 503005.

---

# **5. Abstraction and Information Hiding**

---

Principles of Programming and  
Software Engineering



# What is Abstraction?

- In subsequent weeks, we will learn more about OOP design issues
- One issue is **abstraction**
- **Procedural abstraction**: Specify what to do, not how to do it □ separates the purpose of a method from its implementation
- **Data abstraction**: Specify what you will do to data, not how to do it □ focuses on what operations on the data are to be provided instead of their implementation. More on this when we cover ADT.
- In both cases, we apply **information hiding**
- Ref: Textbook pg 120 – 122

# Procedural Abstraction

Math class

- The API documentation describes **what** **random()** does
  - ❑ What parameters (if any) it takes
  - ❑ What result it returns (if any)
- This provides an interface with the user.
- **How** the method is implemented is hidden from the user.

```
static double
```

```
random()
```

Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.

## random

```
public static double random()
```

Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0. Returned values are chosen pseudorandomly with (approximately) uniform distribution from that range.

When this method is first called, it creates a single new pseudorandom-number generator, exactly as if by the expression

```
new java.util.Random()
```

This new pseudorandom-number generator is used thereafter for all calls to this method and is used nowhere else.

This method is properly synchronized to allow correct use by more than one thread. However, if many threads need to generate pseudorandom numbers at a great rate, it may reduce contention for each thread to have its own pseudorandom-number generator.

### Returns:

a pseudorandom double greater than or equal to 0.0 and less than 1.0.

### See Also:

```
Random.nextDouble()
```

When you write your own methods, you should provide a description of each method like this.

# Summary

- We revisit a few classes ([Scanner](#), [String](#), [Math](#)) and learn a few new ones ([DecimalFormat](#), [Random](#), wrapper classes, [Point](#))
- We discuss some basic OOP features/concepts such as **modifiers**, **class and instance methods**, **constructors** and **overloading**.
- Today, we focus on using classes provided by API as a user.
- Next week, you will become designers to *create* your own classes!

End of file