

Object-Oriented Programming

User Mode



Java

- Introducing API
- Using Java classes
- Basic features/concepts of OOP

1. API: Where you find service classes

1.1 Scanner class (revisit)

1.2 String class (revisit)

1.3 Math class (revisit)

2. OOP concepts (basic)

2.1 Modifiers

2.2 Class vs Instance methods

2.3 Constructors

2.4 Overloading

3. More classes (new)

3.1 `DecimalFormat` class

3.2 `Random` class

3.3 Wrapper classes

3.4 `Point` class

4. Abstraction and Information Hiding

4.1 What is Abstraction?

4.2 Procedural Abstraction

1. API: Where you find service classes

1.1 Scanner class (revisit)

1.2 String class (revisit)

1.3 Math class (revisit)

2. OOP concepts (basic)

2.1 Modifiers

2.2 Class vs Instance methods

2.3 Constructors

2.4 Overloading

3. More classes (new)

3.1 DecimalFormat class

3.2 Random class

3.3 Wrapper classes

3.4 Point class

4. Abstraction and Information Hiding

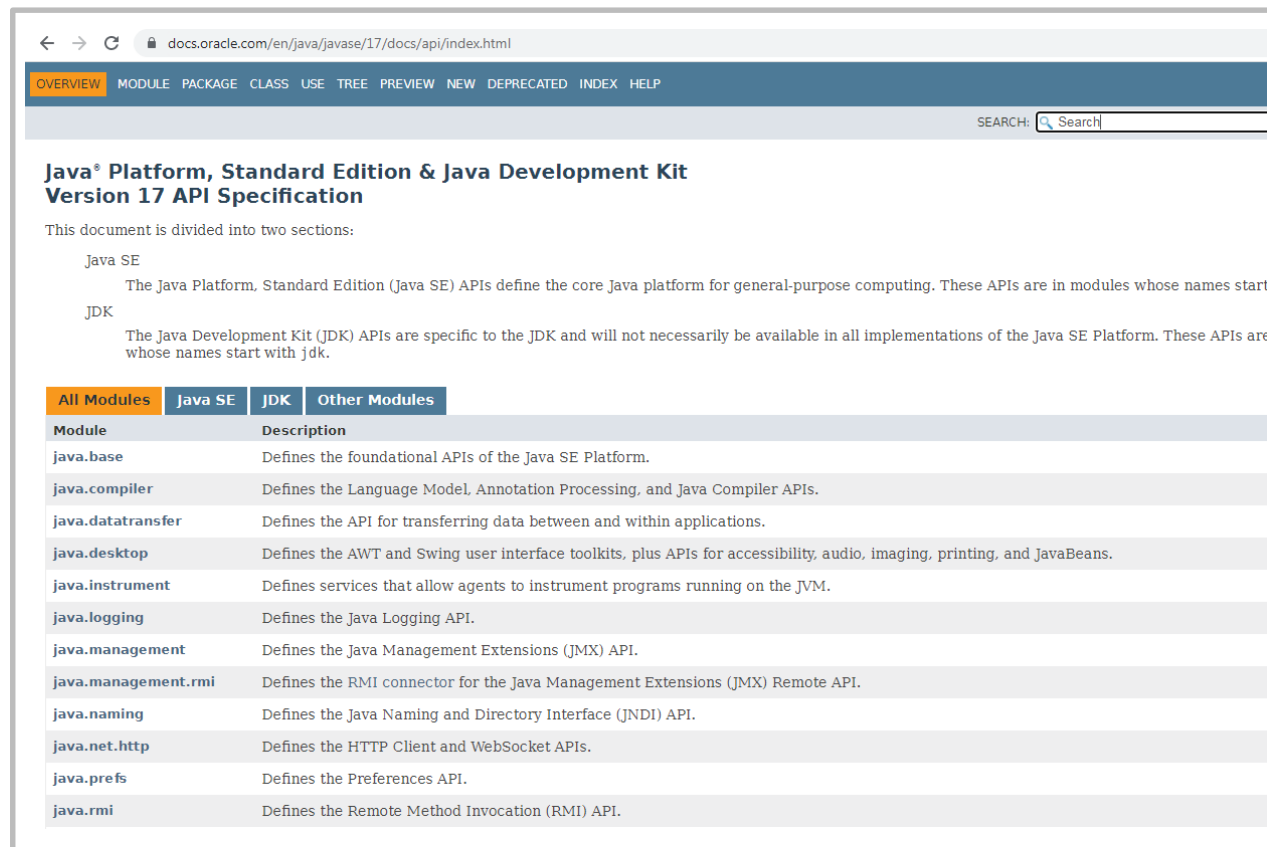
4.1 What is Abstraction?

4.2 Procedural Abstraction

Application Programming Interface
Where you find service classes

API Specification

<http://docs.oracle.com/javase/17/docs/api/>



← → ↻ docs.oracle.com/en/java/javase/17/docs/api/index.html

OVERVIEW MODULE PACKAGE CLASS USE TREE PREVIEW NEW DEPRECATED INDEX HELP

SEARCH:

Java® Platform, Standard Edition & Java Development Kit Version 17 API Specification

This document is divided into two sections:

Java SE

The Java Platform, Standard Edition (Java SE) APIs define the core Java platform for general-purpose computing. These APIs are in modules whose names start with `java.`

JDK

The Java Development Kit (JDK) APIs are specific to the JDK and will not necessarily be available in all implementations of the Java SE Platform. These APIs are whose names start with `jdk.`

All Modules	Java SE	JDK	Other Modules
Module	Description		
<code>java.base</code>	Defines the foundational APIs of the Java SE Platform.		
<code>java.compiler</code>	Defines the Language Model, Annotation Processing, and Java Compiler APIs.		
<code>java.datatransfer</code>	Defines the API for transferring data between and within applications.		
<code>java.desktop</code>	Defines the AWT and Swing user interface toolkits, plus APIs for accessibility, audio, imaging, printing, and JavaBeans.		
<code>java.instrument</code>	Defines services that allow agents to instrument programs running on the JVM.		
<code>java.logging</code>	Defines the Java Logging API.		
<code>java.management</code>	Defines the Java Management Extensions (JMX) API.		
<code>java.management.rmi</code>	Defines the RMI connector for the Java Management Extensions (JMX) Remote API.		
<code>java.naming</code>	Defines the Java Naming and Directory Interface (JNDI) API.		
<code>java.net.http</code>	Defines the HTTP Client and WebSocket APIs.		
<code>java.prefs</code>	Defines the Preferences API.		
<code>java.rmi</code>	Defines the Remote Method Invocation (RMI) API.		



Previous lectures:

Scanner class

String class

Math class

And from now on, many many more...

- API documentation
 - <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Scanner.html>
- For reading input
- Import `java.util.Scanner`

`next()`
`nextDouble()`
`nextInt()`
`nextLine()`
...

`hasNext()`
`hasNextDouble()`
`hasNextInt()`
`hasNextLine()`
...

Note Java naming convention
Method names – **lowerCamelCase**

<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.

TestScanner.java

```
1 import java.util.*;
2
3 public class TestScanner {
4
5     public static void main(String[] args) {
6
7         Scanner sc = new Scanner(System.in);
8
9         // Comparing nextLine() and next()
10        System.out.print("Enter name1: ");
11        String name1 = sc.nextLine();
12        System.out.println("name1 entered is '" + name1 + "'.");
13
14        System.out.print("Enter name2: ");
15        String name2 = sc.next();
16        System.out.println("name2 entered is '" + name2 + "'.");
```

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


TestScanner.java

```
16     sc.nextLine();           // to skip the rest of the line after the next() method
17                               // captured the first word of the second name
18
19     // Using nextInt() and hasNextInt()
20     System.out.println("Enter integers, terminate with control-d:");
21
22     int num;
23     int sum = 0;
24     while (sc.hasNextInt()) {
25         num = sc.nextInt();
26         System.out.println("Integer read: " + num);
27         sum += num;
28     }
29
30     System.out.println("Sum = " + sum);
31 }
32 }
```

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

- API documentation
 - <https://docs.oracle.com/en/java/javase/17/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	<code>indexOf(String str)</code>	Returns the index within this string of the first occurrence of the specified substring.
int	<code>indexOf(String str, int fromIndex)</code>	Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
String	<code>intern()</code>	Returns a canonical representation for the string object.
boolean	<code>isBlank()</code>	Returns true if the string is empty or contains only white space codepoints, otherwise false.
boolean	<code>isEmpty()</code>	Returns true if, and only if, <code>length()</code> is 0.
static String	<code>join(CharSequence delimiter, CharSequence... elements)</code>	Returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
static String	<code>join(CharSequence delimiter, Iterable<? extends CharSequence> elements)</code>	Returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
int	<code>lastIndexOf(int ch)</code>	Returns the index within this string of the last occurrence of the specified character.
int	<code>lastIndexOf(int ch, int fromIndex)</code>	Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
int	<code>lastIndexOf(String str)</code>	Returns the index within this string of the last occurrence of the specified substring.

TestString.java

```
1 public class TestString {
2     public static void main(String[] args) {
3         String text = new String("I'm studying CS1020."); // or String text = "I'm studying CS1020.";
4                                                         // We will explain the difference later.
5
6         System.out.println("text: " + text);
7         System.out.println("text.length() = " + text.length());
8         System.out.println("text.charAt(5) = " + text.charAt(5)); // t
9         System.out.println("text.substring(5, 8) = " + text.substring(5, 8)); // tud
10        System.out.println("text.indexOf(\"in\") = " + text.indexOf("in"));
11
12        String newText = text + "How about you?";
13        newText = newText.toUpperCase();
14        System.out.println("newText: " + newText);
15        if (text.equals(newText)) {
16            System.out.println("text and newText are equal.");
17        } else {
18            System.out.println("text and newText are not equal.");
19        }
20    }
21 }
```

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').

newText = newText.toUpperCase();
converts characters in **newText** to uppercase.

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



- 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)

TestString.java

```
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 strings:
Hello world!
Hello world!
```

(What will be printed?)

- API documentation

- <https://docs.oracle.com/en/java/javase/17/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 θ from the conversion of rectangular coordinates (x, y) to polar coordinates (r, θ).
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 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.

- Here's another demo.

TestMath2.java

```
1 import java.util.*;
2
3 public class TestMath2 {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6
7         System.out.print("Enter 3 values: ");
8         double num1 = sc.nextDouble();
9         double num2 = sc.nextDouble();
10        double num3 = sc.nextDouble();
11
12        System.out.printf("pow(%.2f,%.2f) = %.3f\n", num1, num2, Math.pow(num1, num2));
13        System.out.println("Largest = " + Math.max(Math.max(num1, num2), num3));
14
15        System.out.println("Generating 5 random values:");
16        for (int i = 0; i < 5; i++) {
17            System.out.println(Math.random());
18        }
19    }
20 }
```

```
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
```

1. API: Where you find service classes

1.1 Scanner class (revisit)

1.2 String class (revisit)

1.3 Math class (revisit)

2. OOP concepts (basic)

2.1 Modifiers

2.2 Class vs Instance methods

2.3 Constructors

2.4 Overloading

3. More classes (new)

3.1 DecimalFormat class

3.2 Random class

3.3 Wrapper classes

3.4 Point class

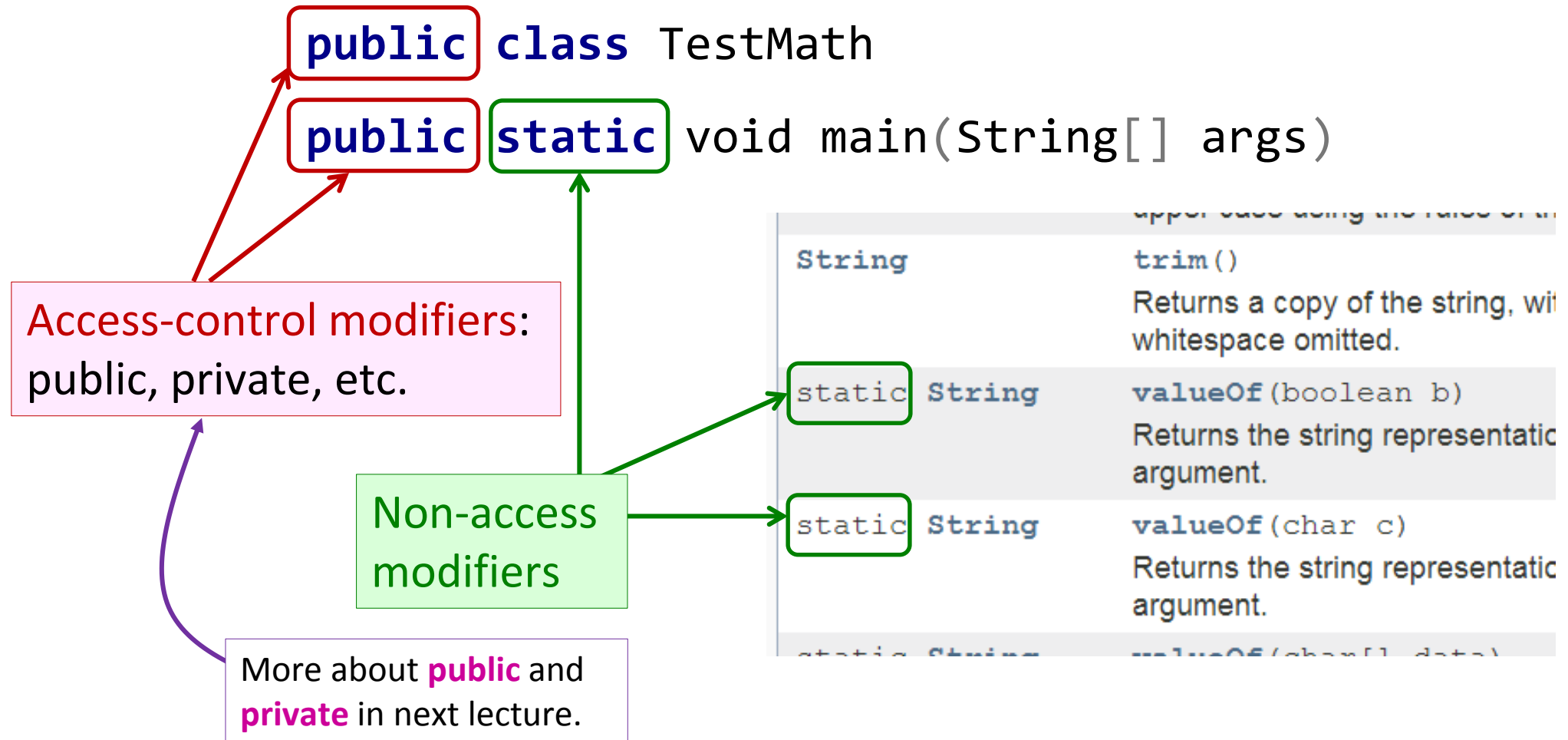
4. Abstraction and Information Hiding

4.1 What is Abstraction?

4.2 Procedural Abstraction

What makes Java object-oriented?

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



String class

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

- 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.

Math class

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

Scanner class

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

String class

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

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

Math
class

■ 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

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

■ Calling a class method

Precede method with the class name

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

```
1 public class Exercise {  
2     public static double volumeCone(double rad, double ht) {  
3         return Math.PI * rad * rad * ht / 3.0;  
4     }  
5  
6     public static void main(String[] args) {  
7         . . .  
8  
9         double vol = volumeCone(radius, height);  
10        /* Alternatively:  
11        * double vol = Exercise.volumeCone(radius, height);  
12        */  
13    }  
14 }
```

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

- Calling a **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)**.

- 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.

- What does `str` contain after the following statement?

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

- 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**.

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

Note the keyword **new**

Constructor and Description
Scanner (File source) Constructs a new <code>Scanner</code> that produces values scanned from the specified file.

Constructor and Description
String () Initializes a newly created <code>String</code> object so that it represents an empty character sequence.
String (String original) Initializes a newly created <code>String</code> object so that it represents the same sequence of characters

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

- 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)

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

Math
class

<code>static double</code>	<code>abs(double a)</code> Returns the absolute value of a <code>double</code> value.
<code>static float</code>	<code>abs(float a)</code> Returns the absolute value of a <code>float</code> value.
<code>static int</code>	<code>abs(int a)</code> Returns the absolute value of an <code>int</code> value.
<code>static long</code>	<code>abs(long a)</code> Returns the absolute value of a <code>long</code> value.

Overloaded methods

String class

- 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.

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

Constructor and Description

<code>String()</code>	Initializes a new <code>String</code> object with an empty character sequence.
<code>String(String original)</code>	Initializes a newly created <code>String</code> object so that it represents the same sequence of characters as the <code>String</code> object <code>original</code> .

Overloaded constructors

1. API: Where you find service classes

- 1.1 Scanner class (revisit)
- 1.2 String class (revisit)
- 1.3 Math class (revisit)

2. OOP concepts (basic)

- 2.1 Modifiers
- 2.2 Class vs Instance methods
- 2.3 Constructors
- 2.4 Overloading

3. More classes (new)

- 3.1 DecimalFormat class
- 3.2 Random class
- 3.3 Wrapper classes
- 3.4 Point class

4. Abstraction and Information Hiding

- 4.1 What is Abstraction?
- 4.2 Procedural Abstraction

Many classes in Java API!

- 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

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: "'o'clock'".

TestDecimalFormat.java

```
1  import java.text.DecimalFormat;
2
3  public class TestDecimalFormat {
4
5      public static void main(String[] args) {
6          DecimalFormat df1 = new DecimalFormat("0.000"); // 3 dec. pl.
7          DecimalFormat df2 = new DecimalFormat("#.###");
8          DecimalFormat df3 = new DecimalFormat("0.00%");
9
10         System.out.println("PI = " + df1.format(Math.PI));
11         System.out.println("12.3 formatted with \"0.000\" = "
12                             + df1.format(12.3));
13         System.out.println("12.3 formatted with \"#.###\" = "
14                             + df2.format(12.3));
15         System.out.println("12.3 formatted with \"0.00%\" = "
16                             + df3.format(12.3));
17     }
18 }
```

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.

- 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

```
static double random()
```

Returns a double 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

■ 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.

- Some methods in `Random` class

<code>double</code>	<code>nextDouble()</code> Returns the next pseudorandom, uniformly distributed <code>double</code> value between <code>0.0</code> and <code>1.0</code> from this random number generator's sequence.
<code>float</code>	<code>nextFloat()</code> Returns the next pseudorandom, uniformly distributed <code>float</code> value between <code>0.0</code> and <code>1.0</code> 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 <code>0.0</code> and standard deviation <code>1.0</code> 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 <code>0</code> (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.

TestRandom.java

```
1  import java.util.Random;
2
3  public class TestRandom {
4
5      public static void main(String[] args) {
6          // To generate a random integer in [51,70]
7          // using Math.random() and Random's nextInt()
8          int num1 = (int)(Math.random() * 20) + 51;
9          System.out.println("num1 = " + num1);
10
11         Random random = new Random();
12         int num2 = random.nextInt(20) + 51;
13         System.out.println("num2 = " + num2);
14     }
15 }
```

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.

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

- 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>
and others...	

- 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
```

```
String str = Integer.toString(567); // str contains "567"
```

- Look up the API documentation and explore the wrapper classes on your own

- 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

The screenshot shows the Java Platform Standard Ed. 7 documentation for the **Class Point**. The left sidebar lists packages and classes, with **Point** selected. The main content area shows the class hierarchy, implemented interfaces, and the start of the class declaration.

Java™ Platform
Standard Ed. 7

Overview Package **Class** Use Tree Deprecated Ind

Prev Class Next Class Frames No Frames

Summary: Nested | Field | Constr | Method Detail: Field | Cons

java.awt

Class Point

java.lang.Object
 java.awt.geom.Point2D
 java.awt.Point

All Implemented Interfaces:

Serializable, Cloneable

```
public class Point
```

- 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	x The X coordinate of this <code>Point</code> .
int	y The Y coordinate of this <code>Point</code> .

- 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)

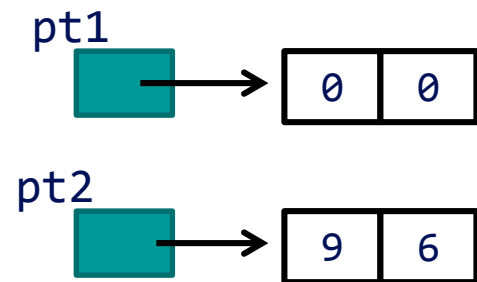
Constructs and initializes a point at the specified (x, y) location in the coordinate space.

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)
```



■ Methods in **Point** class

Methods	
Modifier and Type	Method and Description
boolean	equals (Object obj) Determines whether or not two points are equal.
Point	getLocation () Returns the location of this point.
double	getX () Returns the X coordinate of this Point2D in double precision.
double	getY () Returns the Y coordinate of this Point2D in double precision.
void	move (int x, int y) Moves this point to the specified location in the (x, y) coordinate plane.
void	setLocation (double x, double y) Sets the location of this point to the specified double coordinates.
void	setLocation (int x, int y) Changes the point to have the specified location.
void	setLocation (Point p) Sets the location of the point to the specified location.
String	toString () Returns a string representation of this point and its location in the (x, y) coordinate space.
void	translate (int dx, int dy) 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).

TestPoint.java

```
1  import java.util.*;
2  import java.awt.*;
3
4  public class TestPoint {
5
6      public static void main(String[] args) {
7          Scanner sc = new Scanner(System.in);
8          System.out.print("Enter x and y: ");
9          int xCoord = sc.nextInt();
10         int yCoord = sc.nextInt();
11
12         Point pt = new Point(xCoord, yCoord);
13         System.out.println("x-coordinate is " + pt.getX());
14         System.out.println("y-coordinate is " + pt.y);
15
16         System.out.println("The point created is " + pt);
17         // or: System.out.println("The ... is " + pt.toString());
18     }
19 }
```

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.

- Accessing an object before it is created

WRONG!

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

pt

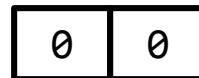


The Point object does not even exist!

RIGHT!

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

pt





- 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 **practise 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.

1. API: Where you find service classes

- 1.1 Scanner class (revisit)
- 1.2 String class (revisit)
- 1.3 Math class (revisit)

2. OOP concepts (basic)

- 2.1 Modifiers
- 2.2 Class vs Instance methods
- 2.3 Constructors
- 2.4 Overloading

3. More classes (new)

- 3.1 DecimalFormat class
- 3.2 Random class
- 3.3 Wrapper classes
- 3.4 Point class

4. Abstraction and Information Hiding

- 4.1 What is Abstraction?
- 4.2 Procedural Abstraction

Principles of Programming and Software Engineering

- 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**

- 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.

Math class

```
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.

- 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 lecture, you will become [designers](#) to *create* your own classes!

- Important that you explore on your own after lecture!
- OOP involves many concepts, too many to be covered in one or two lectures.
- Hence, you cannot expect to learn everything in just one sitting. You probably need to re-visit the topics/concepts over and over again.
- Additional materials may be introduced in tutorials/labs.
- Attempt the practice exercises.
 - Many of the practice exercises are simple exercises to test your understanding of the very basic – **must do them!**

Thank you!

