ĐẠI HỌC TÔN ĐỨC THẮNG
TON DUC THANG UNIVERSITY

Data Structures and Algorithms I

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.

Recording of modifications

- Course website address is changed to http://sakai.it.tdt.edu.vn
- Slides “Practice Exercises” (#22 and #49) are eliminated.
- Course codes cs1010, cs1020, cs2010 are placed by 501042, 501043, 502043 respectively.

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)

IT-TDT Sakai → 501043 website → Lessons

- http://sakai.it.tdt.edu.vn

Outline (1/2)

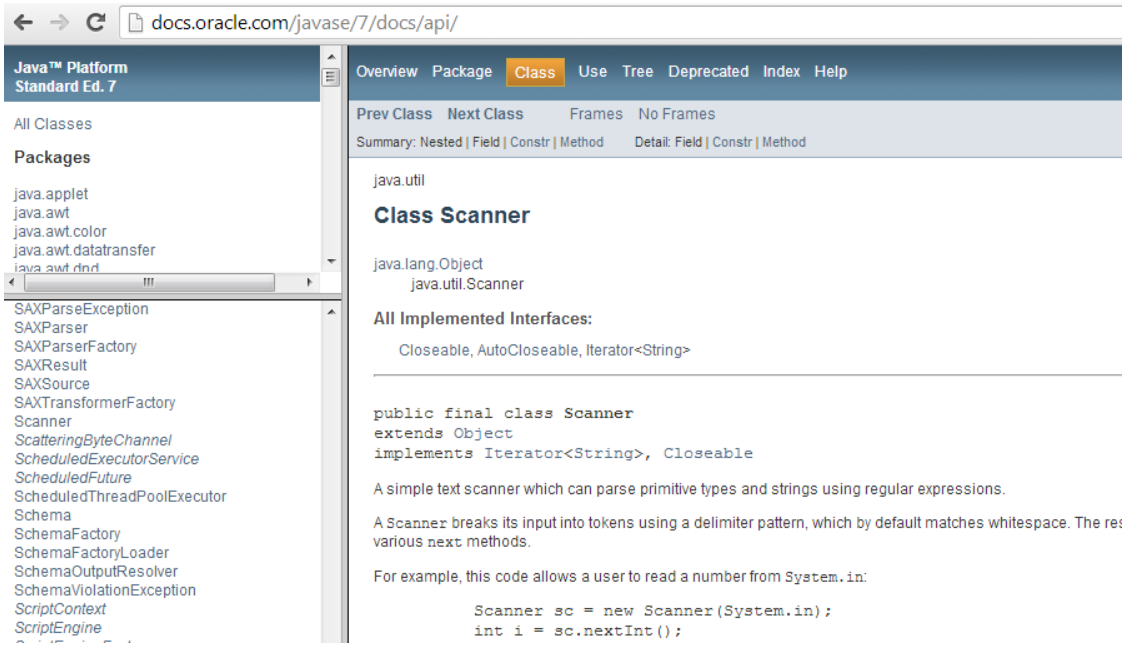
1. Recapitulation
2. API: Where you find service classes
   - 2.1 Scanner class (revisit)
   - 2.2 String class (revisit)
   - 2.3 Math class (revisit)
3. OOP concepts (basic)
   - 3.1 Modifiers
   - 3.2 Class vs Instance methods
   - 3.3 Constructors
   - 3.4 Overloading

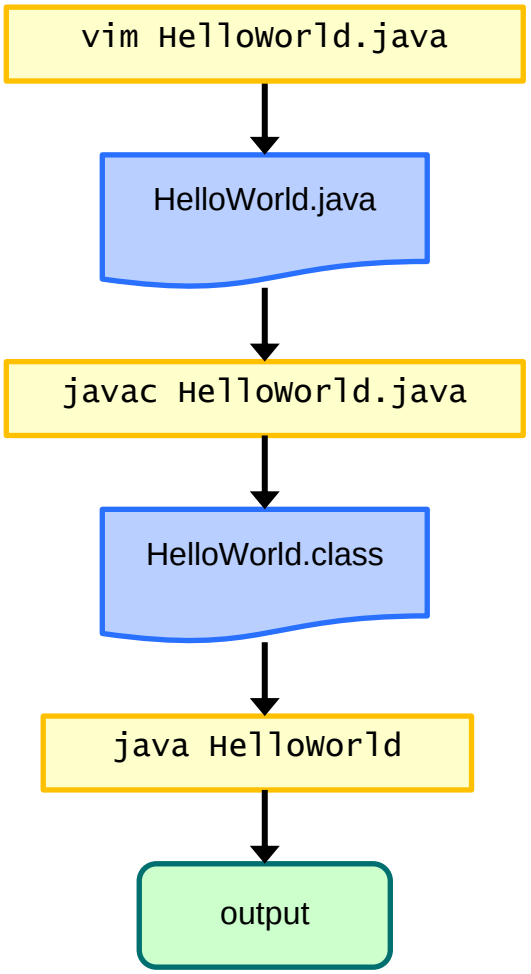
Outline (2/2)

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





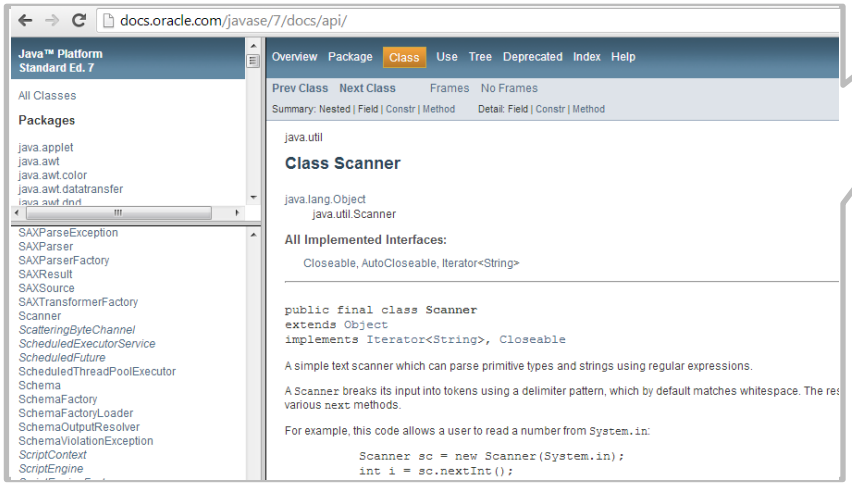
2. API (Revisit)

Application Programming Interface – Where you find service classes

Java Programmer

API Specification

http://docs.oracle.com/javase/7/docs/api/



Last week:

Scanner class

String class

Math class

And from now on,
many many more…

Scanner Class: Reading Inputs

- API documentation
  - http://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html
- For reading input
- Import java.util.Scanner

Note Java naming convention
Method names – lowerCamelCase

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

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

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

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);
   }
}
```

What is this for?
Attend lecture for explanation!

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

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
  - http://docs.oracle.com/javase/7/docs/api/java/lang/String.html
- Import java.lang.String (optional)
- Ubiquitous; Has a rich set of methods

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

And many more…

int	**indexOf**(int ch) Returns the index within this string of the first occurrence of the specified character.
int	**indexOf**(int ch, int fromIndex) Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
int	**indexOf**(String str) Returns the index within this string of the first occurrence of the specified substring.
int	**indexOf**(String str, int fromIndex) Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
String	**intern**() Returns a canonical representation for the string object.
boolean	**isEmpty**() Returns true if, and only if, **length()** is 0.
int	**lastIndexOf**(int ch) Returns the index within this string of the last occurrence of the specified character.
int	**lastIndexOf**(int ch, int fromIndex) Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
int	**lastIndexOf**(String str) Returns the index within this string of the last occurrence of the specified substring.
int	**lastIndexOf**(String str, int fromIndex) Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
int	**length**() Returns the length of this string.
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 501043.");
      // or String text = "I'm studying 501043.";
      // 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.");
   }
}
```

String Class: Demo (2/2)

Outputs

Explanations

```
text: I'm studying 501043.
```

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

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

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

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

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

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

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

`indexOf("in")` returns the …?

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

```
newText: I'M STUDYING 501043.HOW ABOUT YOU?
```

The + operator is string concatenation.

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

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

String Class: Comparing strings

- As strings are objects, do not use == if you want to check if two strings contain the same text
- Use the equals() method provided in the String class instead (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
  - http://docs.oracle.com/javase/7/docs/api/java/lang/Math.html
- Import java.lang.String (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 *pi*.
static double	**asin**(double a) Returns the arc sine of a value; the returned angle is in the range -*pi*/2 through *pi*/2.
static double	**atan**(double a) Returns the arc tangent of a value; the returned angle is in the range -*pi*/2 through *pi*/2.
static double	**atan2**(double y, double x) Returns the angle *theta* from the conversion of rectangular coordinates (x, y) to polar coordinates (r, *theta*).
static double	**cbrt**(double a) Returns the cube root of a double value.
static double	**ceil**(double a) Returns the smallest (closest to negative infinity) double value that is greater than or equal to the argument and is equal to a mathematical integer.
static double	**copySign**(double magnitude, double sign) Returns the first floating-point argument with the sign of the second floating-point argument.
static float	**copySign**(float magnitude, float sign)

…nd floating-point argument.

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

Math Class: Demo

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

```
import java.util.*;

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

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

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

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

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

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

TestMath2.java

3. OOP Concepts

What makes Java object-oriented?

Modifiers

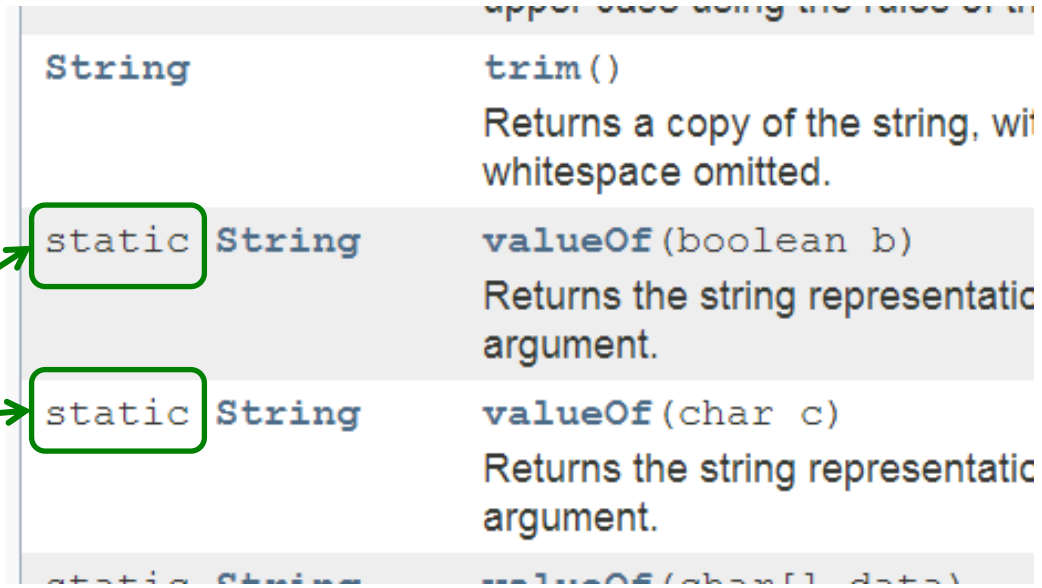
: keywords added to specify the way a class/attribute/method works

```
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, wi whitespace omitted.
static String	valueOf(boolean b) Returns the string representatic argument.
static String	valueOf(char c) Returns the string representatic argument.

Class vs Instance methods (1/4)

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

Math class

`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

- 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)

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

Math class

`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

- 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 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 (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) {
      . . .

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

}
```

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

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:

`static String`	**`valueOf`**`(double d)` Returns the string representation of the `double` argument.
`static String`	**`valueOf`**`(float f)` Returns the string representation of the `float` argument.
`static String`	**`valueOf`**`(int i)` Returns the string representation of the `int` 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
`String``(String original)` Initializes a newly created `String` object so that it represents the same sequence

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. *Need not be quoted in prefix or suffix.*
;	Subpattern boundary	Yes	Separates positive and negative subpatterns
%	Prefix or suffix	Yes	Multiply by 100 and show as percentage
¤ (\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"`.

Example:

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

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%\" = "
                         + df3.format(12.3));
   }
}
```

```
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

`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

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

double	nextDouble() Returns the next pseudorandom, uniformly distributed double value between 0.0 and 1.0 from this random number generator's sequence.
float	nextFloat() Returns the next pseudorandom, uniformly distributed float value between 0.0 and 1.0 from this random number generator's sequence.
double	nextGaussian() Returns the next pseudorandom, Gaussian ("normally") distributed double value with mean 0.0 and standard deviation 1.0 from this random number generator's sequence.
int	nextInt() Returns the next pseudorandom, uniformly distributed int value from this random number generator's sequence.
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.

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

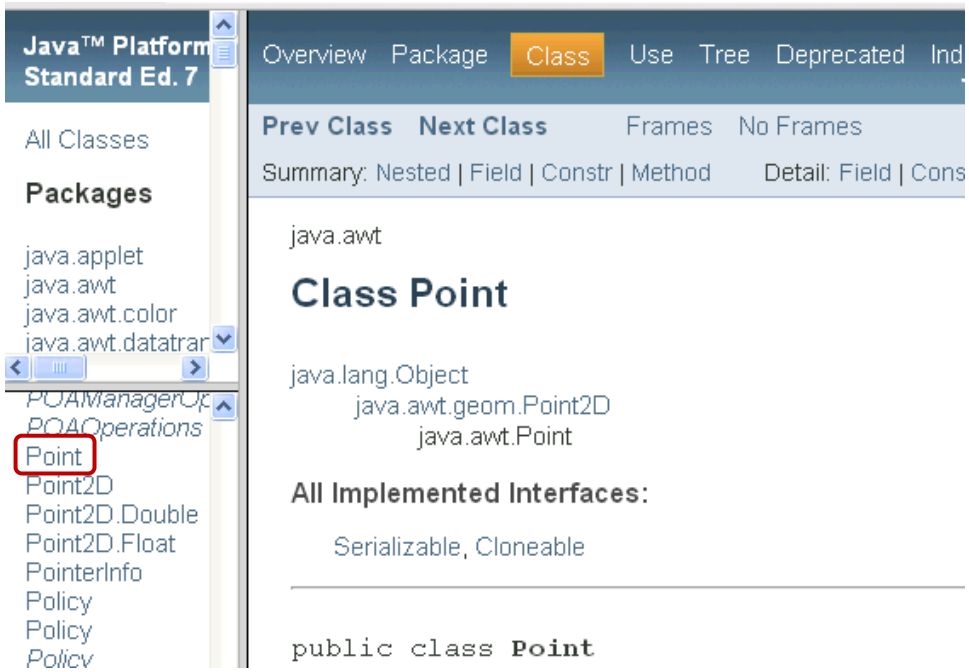
Primitive data type	Wrapper class
`int`	`Integer`
`long`	`Long`
`float`	`Float`
`double`	`Double`
`char`	`Character`
`boolean`	`Boolean`
and others…	

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



Point Class (2/5): Attributes

- The Point class contains 2 attributes
  - Sometimes also called data members
  - In the API documention, 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

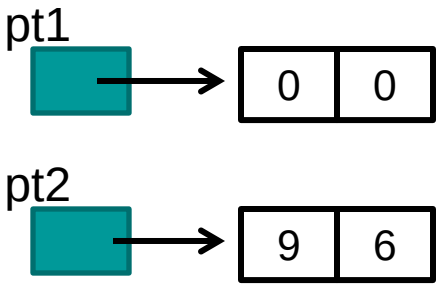
Modifier and Type	Field and Description
`int`	**`x`** The X coordinate of this `Point`.
`int`	**`y`** The Y coordinate of this `Point`.

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`	**`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)`.

Point Class (5/5): Demo

TestPoint.java

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

public class TestP
   public static v
      Scanner sc =

      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
y-coordinate is -7
The point created is java.awt.Point[x=12,y=-7]
```

← *Note:* getX() returns double

To be discussed in next lecture.

Common Mistake

- Accessing an object before it is created

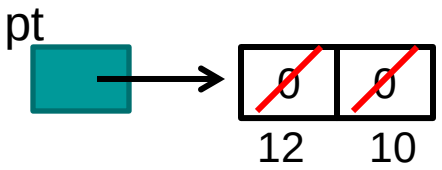
WRONG!

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



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, which include discussion on the IVLE forums.
- 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.
- Like 501042 (or equivalent), you must be prepared to invest time in 501043.

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

- 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.

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!

Advice

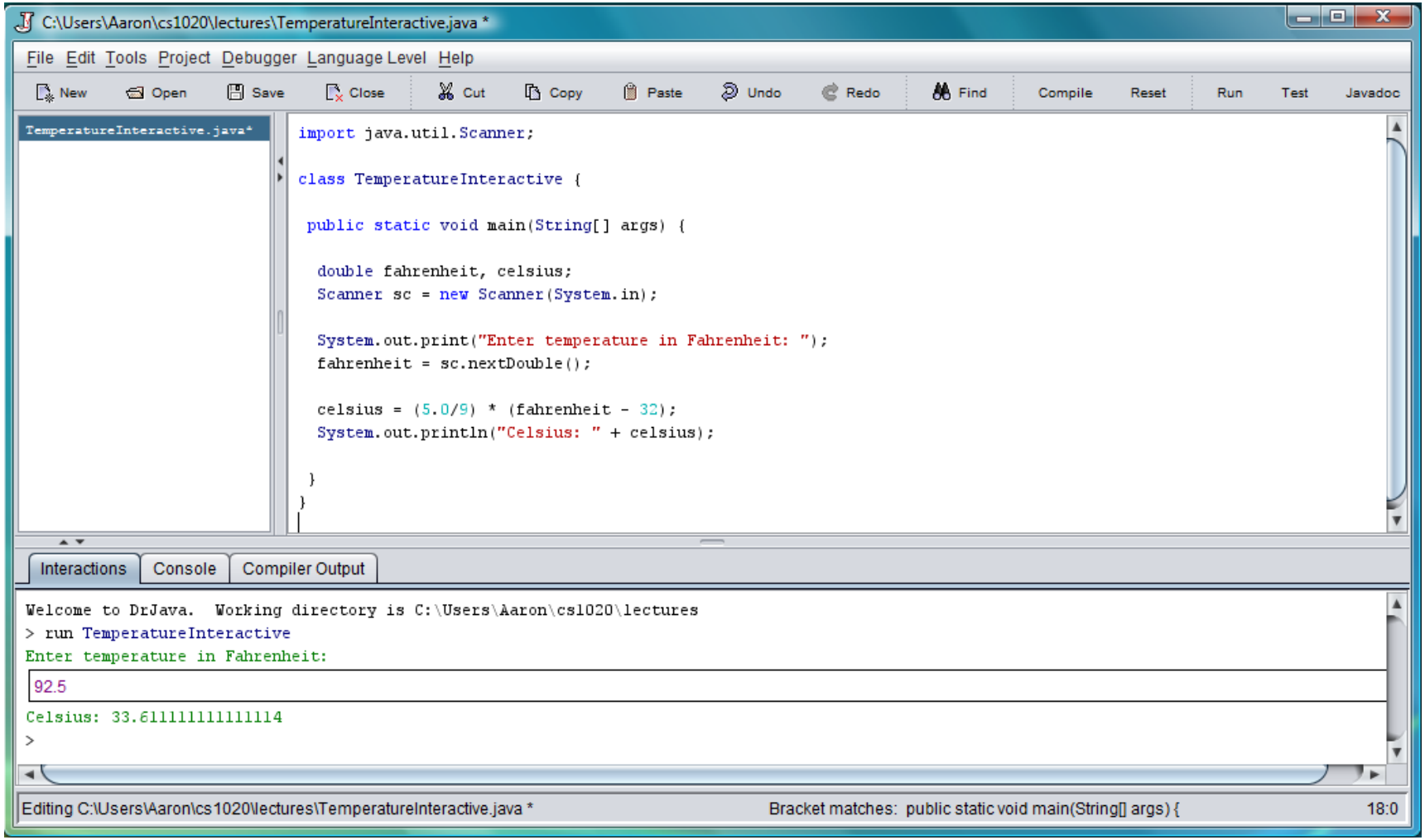
- Important that you explore on your own after lecture!
- OOP involves many concepts, too many to be covered in one or two lectures.
- Hence, you cannot expect to learn everything in just one sitting. You probably need to re-visit the topics/concepts over and over again.
- Additional materials may be introduced in tutorials/labs.
- Attempt the practice exercises and the IVLE self-assessments. They are not graded.
  - Many of the practice exercises are simple exercises to test your understanding of the very basic – must do them!
- Please post your queries on the IVLE forum. Try not to email us, unless your queries are of private nature.

Misc.: Dr Java (1/3)

- http://drjava.sourceforge.net/
- DrJava is a simple IDE you may use for practice on Java programming.
- This is only for your own use. For sit-in labs, you will be given a special UNIX account and you must work in the UNIX environment, using the vim editor. (Non-501042 students please take note.)

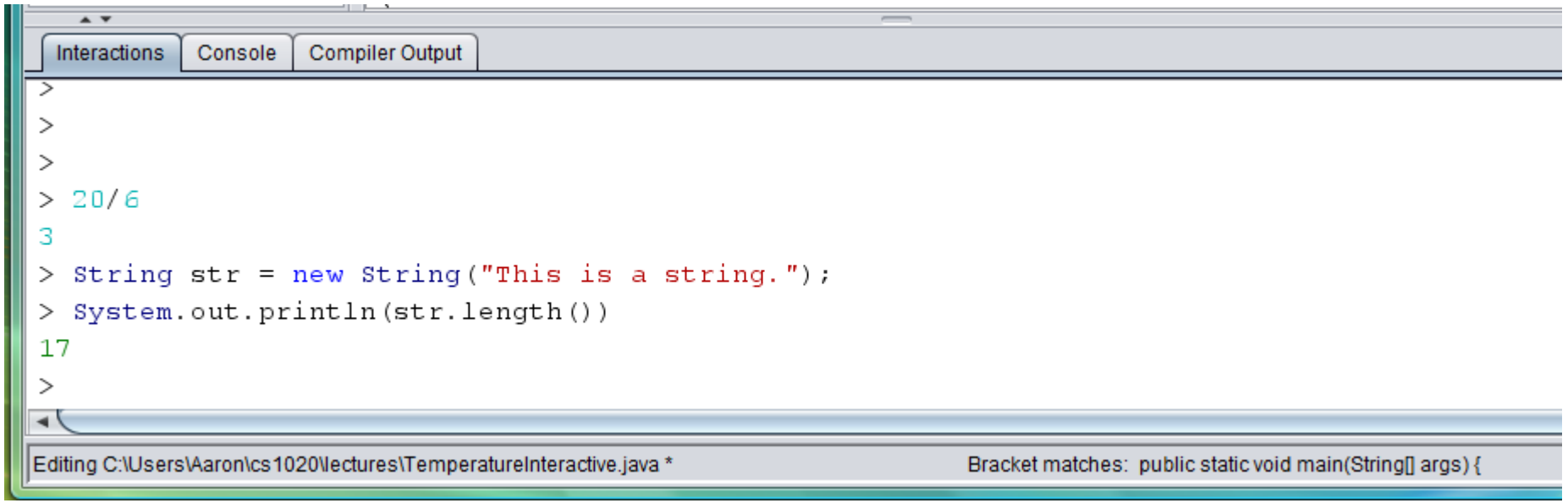
Misc.: Dr Java (2/3)

- Running Week 1's TemperatureInteractive.java



Misc.: Dr Java (3/3)

- You may also type statements directly in the "Interactions" pane
- Good for quick checks



End of file