Computer Creativity

From Processing to Java

Slides courtesy of Dr. Abdallah Mohamed.

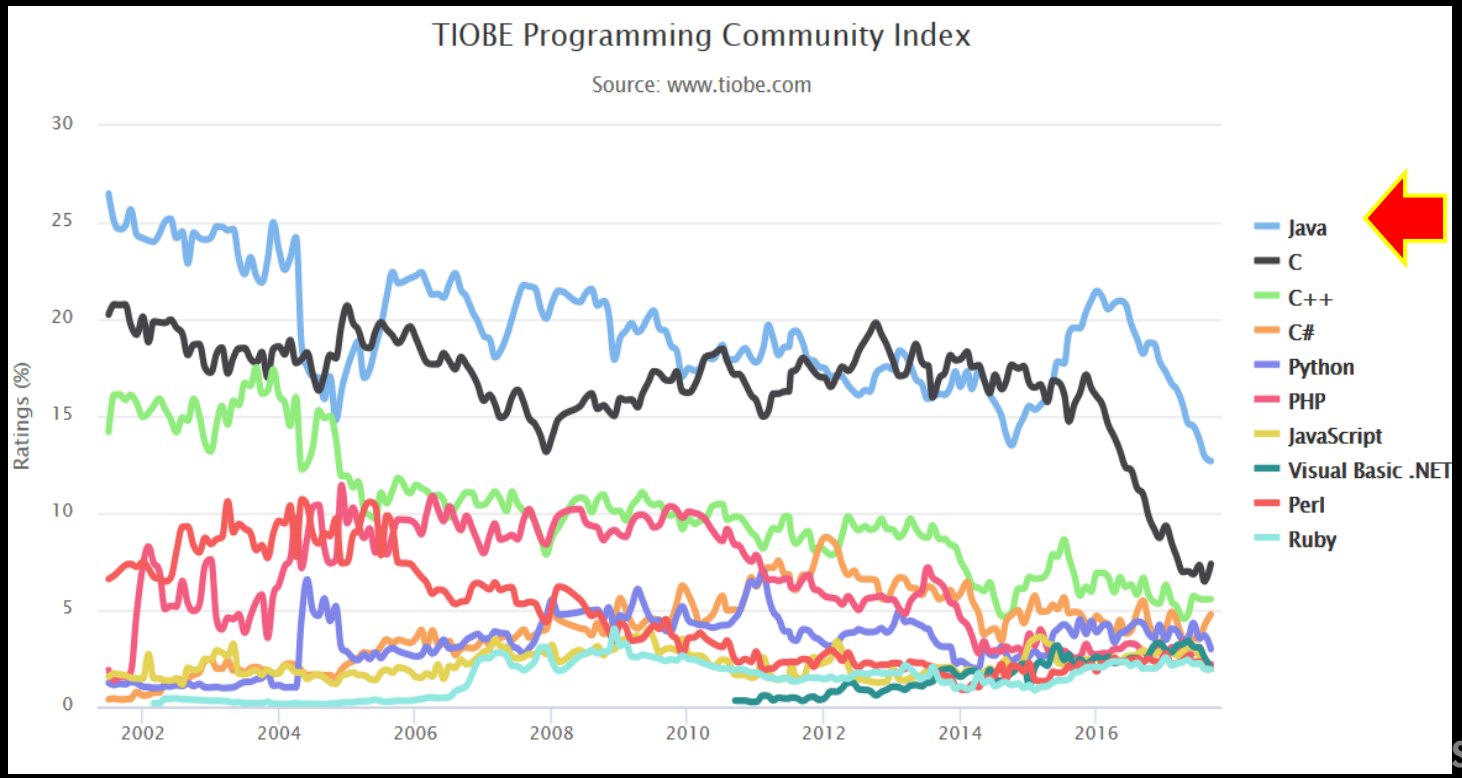
Processing vs. Java

- **Java** is a general-purpose, object-oriented language developed in 1991 by a group led by James Gosling and Patrick Naughton of Sun Microsystems.
- Java allows creating anything and runs on most computers and cell phones.
- **Java is an industry standard, and it is taught in** several UBCO courses, e.g. COSC 111, 121 and 222.

- **Processing** is a programming environment that aims to help create visually oriented applications, such as sketches, animations, and games. Processing is built on Java.

Java

- Major advantages of Java:
  - Can run on almost any type of machine.
  - Popular language for web and system development.
  - Good teaching language because many issues such as memory management are hidden.



Processing vs. Java, *cont'd*

Processing IS Java*

- But it simplifies a lot of complexities for building graphical apps.
- By learning Processing, you have already been learning Java!!!
  - e.g. data types, variables, arrays, selection, loops, OOP, etc.
- Processing code actually compiles into Java JAR file.
  - JAR files are container files, just like zip files, that aggregate different project-related files into one.
- You can easily convert your Processing programs to Java by adding some extra code and libraries.
- `File>Export` (in the PDE) creats
  - (1) executable files (.exe and .jar files)
  - (2) Java source code (.java file)

* Initially Processing was built on java. Later, other "modes" were introduced for other languages such as JavaScript

Example: Exporting Processing to Java

Processing

```
float w = 30.5;
void setup() {
    size(100, 100);
}
void draw() {
    background(100);
    fill(255);
    rect(mouseX,mouseY,w,w);
    w += random(-1,1);
}
```

Java

```
import java.applet.*;
import java.awt.*;
import java.awt.image.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import java.text.*;
import java.util.*;
import java.util.zip.*;

import processing.core.*;

public class Ex1 extends PApplet {
    float w = 30.5f;
    public void setup() {
        size(100, 100);
    }
    public void draw() {
        background(100);
        fill(255);
        rect(mouseX,mouseY,w,w);
        w += random(-1,1);
    }
    ...
}
```

Java libraries

Processing library

Subclass of JApplet

might be a bit differnt on your computer

The `import` statement

- Java comes with a set of built-in classes and functions. These classes are grouped in ***packages***.
  - You can think of a package as a folder on your computer in which Java classes are stored.
- The `import` statement allows you to refer to ***classes*** that are declared outside your program's package.

Syntax:

```
import processing.core.*
```

include the classes from following package (i.e. folder) in my program

the folder that contains the classes to be imported

import ALL classes from processing.core package

The `import` statement, cont'd

- You can choose to import a single class from a package or all classes in that package.

```
import processing.core.*
```

import ALL classes from processing.core package. This means you get to use ***any of them*** in your program

```
import processing.core.PImage
```

import only PImage class from processing.core. This means you cannot use other classes in that package in your program

The `processing.core` Library

- This package has classes with the drawing functions that we have been using in this course
  - E.g. line(), color(), rect(), etc
- These classes are saved on your computer under your Processing folder → core\library\core.jar
- The classes in the above folder are already compiled (they have .class file format)
  - i.e. compiled means transformed from the source-code readable format (.java) to another format (.class).
- The source code (with .java extension) of the above classes can be found on:
  github.com/processing/processing/tree/master/core/src/processing/core

How to Draw in Java?

- Without the `processing.core` library, you won't be able to use most of the drawing functions we used in this course.
- Is there another way to draw in Java?
  - Yes, there are several options for drawing in Java. For example, using JavaFX or Java Swing library.

Examples of some statements in Processing and their Java equiv.

Processing	Java
`stroke(0);`	`g.setColor(Color.black);`
`line(0, 10, 20, 30);`	`g.drawLine(0,10,20,30);`
`void mousePressed(){…}`	`public void mousePressed(MouseEvent e){…}`

`mouseX`
`mouseY`
	`public void mouseMoved(MouseEvent e){`
	`  mouseX = e.getX();  mouseY = e.getY();`
	`}`
	`public oid mouseDragged(MouseEvent e){`
	`  mouseX = e.getX();  mouseY = e.getY();`
	`}`

Console Applications

- Console applications are computer programs that are used only through text-based interfaces.

- The remaining part of this unit we will do two things:
  - **1) Examples**: we will see a few examples that demonstrate console applications in Processing and their Java equivalences.
    - Note that we will not use any of the drawing functions in the processing.core library.
  - **2) Eclipse** which is an IDE that can be used for Java development.
  - **3) More on Java**: we will also see a few additional Java statements that would help you in switch to Java (especially if you are going from COSC123 to COSC 121). Namely, we are going to look at:
    - a) how to read input from the console in Java.
    - b) a few more Java built-in functions

1) Examples of Console Apps

(Processing vs. Java)

Example 1: Compute the Area of a Circle

- This program computes and displays the area of a circle with radius = 5.
- Processing:

Ex1

```
1 double radius, area;
2 radius = 20;
3 area = radius * radius * 3.14159;
4 print ("Area: " + area);
```

- Java :
  - main() is the first function executed in Java.
  - `System.out.println()` is the equivalent function in Java for Processing's `print()` function

*Ex1.java

```
1 public class Ex1 {
2     public static void main(String[] args) {
3         double radius, area;
4         radius = 20;
5         area = radius * radius * 3.14159;
6         System.out.println("Area: " + area);
7     }
8 }
```

Example 2: Find the Sum of an Array

- This program creates and initializes an integer array, then it displays the sum of all array elements.
- Processing:

Ex1

```
1 int[] list = {3, 5, 2, 1};
2 int sum = 0;
3 for(int i = 0; i<list.length; i++)
4   sum += list[i];
5 print(sum);
```

- Java:

Ex1.java

```
1 public class Ex1 {
2     public static void main(String[] args) {
3         int[] list = {3, 5, 2, 1};
4         int sum = 0;
5         for(int i = 0; i<list.length; i++)
6           sum += list[i];
7         System.out.println(sum);
8     }
9 }
```

Example 3: OOP

- This program creates a class Circle then creates a circle object and print out its radius and area twice.
  - Note that PI constant in Processing is replaced by Math.PI in Java

Processing

Ex1

```
1  void setup(){
2    Circle c = new Circle();
3    println("Radius:"+c.r+", Area="+c.getArea());
4    c.setRadius(3);
5    println("Radius:"+c.r+", Area="+c.getArea());
6  }
7
8  class Circle{
9    double r;
10   Circle(){r = 1.0;}
11   Circle(double a){r = a;}
12   void setRadius(double a){r = a;}
13   double getArea(){return r*r*PI;}
14   double getPerimeter(){return 2*r*PI;}
15 }
```

Java

*Ex1.java

```
1  public class Ex1 {
2      public static void main(String[] args) {
3          Circle c = new Circle();
4          System.out.println("Radius:"+c.r+",Area="+c.getArea());
5          c.setRadius(3);
6          System.out.println("Radius:"+c.r+",Area="+c.getArea());
7      }
8  }
9
10 class Circle{
11     double r;
12     Circle(){r = 1.0;}
13     Circle(double a){r = a;}
14     void setRadius(double a){r = a;}
15     double getArea(){return r*r*Math.PI;}
16     double getPerimeter(){return 2*r*Math.PI;}
17 }
```

2) Eclipse

Eclipse IDE

- It is possible to write Java programs using any text editor and compile them using the Java compiler.

- An ***integrated development environment (IDE)*** makes it easier to write code, find errors, and run your programs.

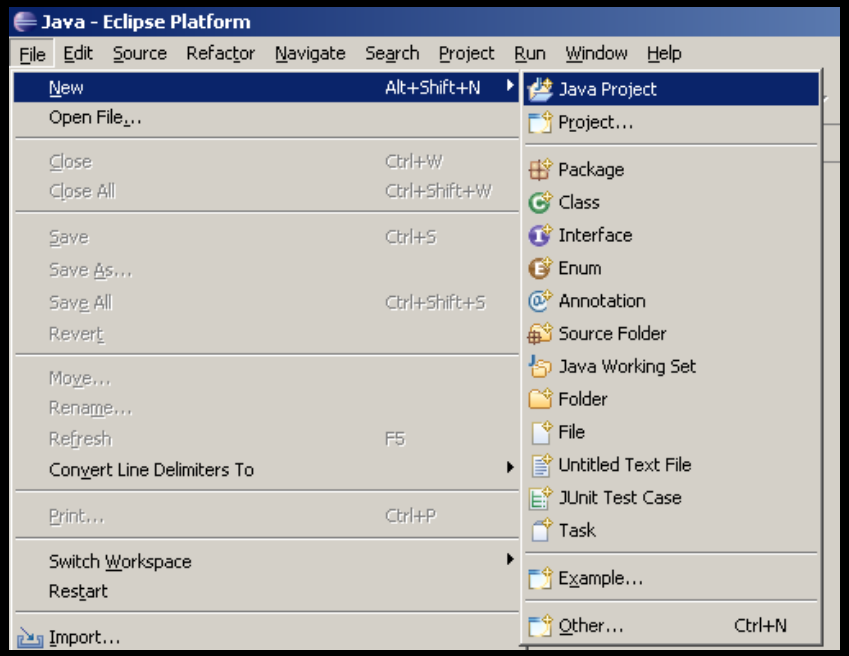
- ***Eclipse*** is one of the most common environment IDEs for Java.
  - Eclipse is a generic, extensible development environment that can be used for Java and other languages.
  - Eclipse makes coding easier with automatic error checking, code completion, and source debugging.
  - Eclipse will **NOT** make it easier to figure out **WHAT** to write, but it will make **HOW** to write it easier.

Eclipse Initial Setup
Creating a Workspace and a Project

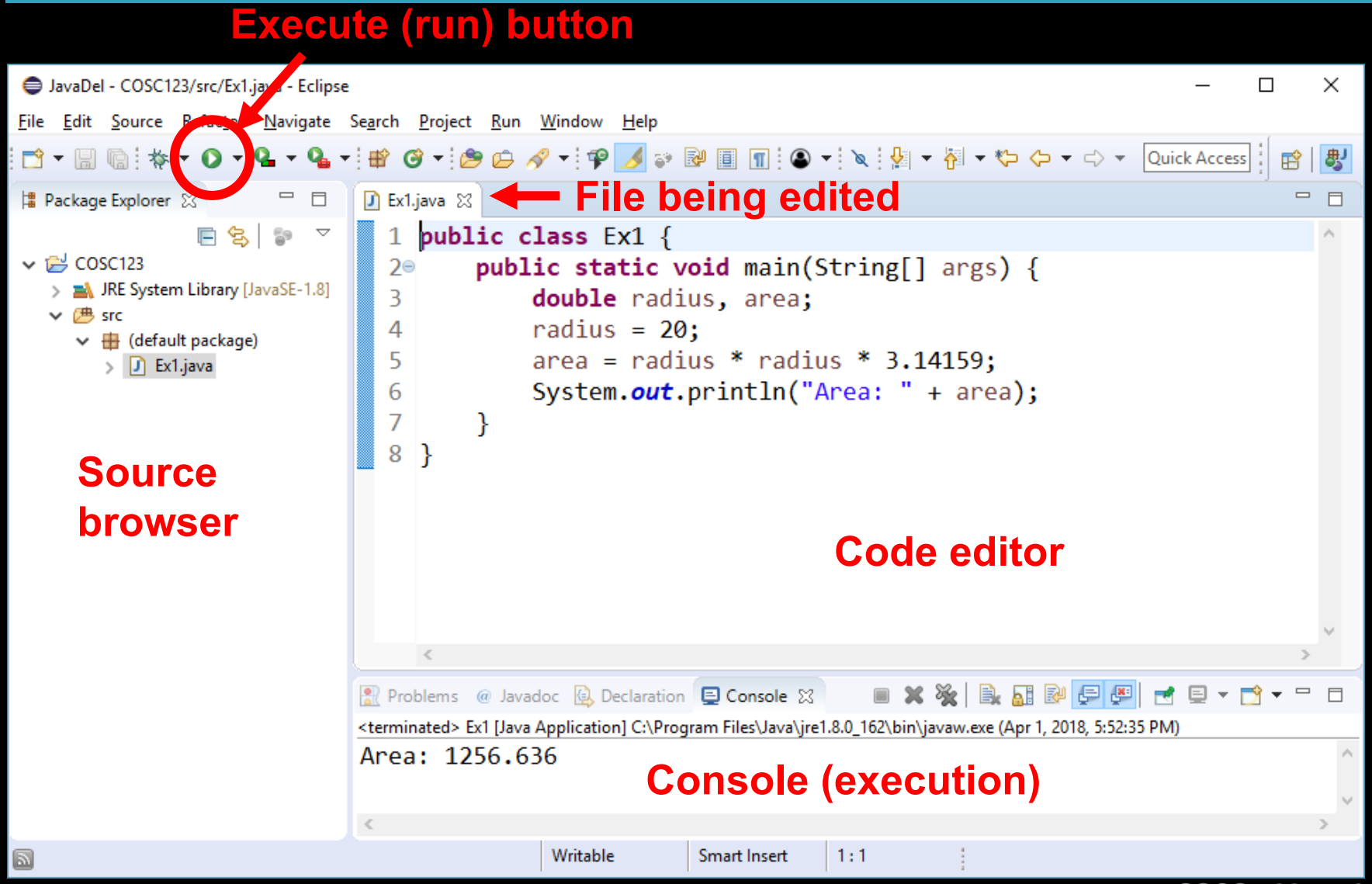
- A ***workspace*** is the place where Eclipse will store all of your projects.
  - You will be prompted for your workspace on start up if you have not selected one.

- A ***project*** is a group of program files for some purpose. We will create a sample project called **`cosc123`**. You will also create projects for each assignment.
  - Give the project a name and click finish. Ignore all options for now.

Create a New Project using
`File->New->Java Project`



Eclipse Main Screen



Installing Eclipse on Your Computer

- In order to install and use Eclipse on your machine, you need to:
  - 1) install JDK (Java Development Kit), which includes a complete set of tools for developing, debugging, and running Java applications.
    - To install it, google the term JDK, then install the Java Platform Standard Edition.
  - 2) install Eclipse
    - To install it, google the term Eclipse and install the “Eclipse for Java Developers” edition.

Aside: Processing in Eclipse

To use processing.core library in Eclipse, do the following:

1. Create a new project and a class
   `File>New>Java Project`, then `File>New>Class`
2. Include the Processing Core in your project
   a) Choose your project then select
      `File>Import>General>File System`
   b) Browse to your processing folder /core/library, then choose at least core.jar
   c) Right click on `core.jar` file in your project and choose
      `Build Path>Add to Build Path`
   d) The file should be added into a new section called “Referenced Libraries” under your project.
3. Write your class using the structure on the next slide.

Aside: Processing in Eclipse, cont'd

```
package p;
import processing.core.*;

public class Ex1 extends PApplet{
    public void settings() {
      size(w,h);                    //put size method here
    }
    public void setup(){

    }
    public void draw() {

    }
    public static void main(String[] args) {
      PApplet.main("p.Ex1");  //String must match package.class name
    }
}
```

Aside: Project with Multiple Classes in Eclipse

The main class, i.e. the one with the public method similar to previous slide, can create instances of any other classes defined outside it. However, you must pass a reference of the main class to the other classes and then use that reference with any processing API calls.

```
package test;
import processing.core.*;
public class Main extends PApplet {
  Ball ball;
  public void settings() {
    size(200,200);
  }
  public void setup() {
    ball = new Ball(this);
  }
  public void draw() {
    background(0);
    ball.display();
  }

  public static void main(String[] args){
    PApplet.main("test.Main");
  }
}
```

```
package test;

public class Ball {
  Main parent;  //parent PApplet where all drawings go
  int w, h, x, y;
  public Ball(Main parent)  {//ball knows about its parent
    this(parent.width/2, parent.height/2,50,50,parent);
  }
  public Ball(int x,int y,int w,int h,Main parent) {
    this.x = x; this.y = y;
    this.w = w; this.h = h;
    this.parent = parent;
  }

  void display() {
    parent.fill(255,0,0);//draw on parent PApplet
    parent.ellipse(x, y, w, h);
  }
}
```

3) More on Java (optional reading)

Read if you are using COSC 123
as a prerequisite to COSC 121.

More on Java

- The next few slides introduce more topics from Java language.
- The topics and functions discussed can be perfectly used in Processing (since processing is in fact Java).
- However, these topics are not part of the final exam of COSC123. However, should you decide to use only COSC123 as a prerequisite for COSC121, you must read this section.
- Note that you have already learned all what is needed for COSC121, but these extra slides include the topics that were not discussed in our COSC123 course.

- Note: the following remaining slides are taken from COSC111 course.

Primitive Data Types

	Java	Size in memory	Range
whole numbers	byte	8 bits	-2^7 to 2^7-1 (-128 to 127)
	short	2 bytes	-2^{15} to $2^{15}-1$ (-32768 to 32767)
	int	4 bytes	-2^{31} to $2^{31}-1$
	long	8 bytes	-2^{63} to $2^{63}-1$
real numbers	float	4 bytes	e.g. 17.345f
	double	8 bytes	e.g. 12452.212 (more accurate)
characters	**char**	2 bytes	e.g. 'a', '1‘and '?'
boolean	**boolean**	1 byte	true or false

Formatting Console Output: `printf`

You can use the **System.out.printf** *method to display formatted output on the console:*

```
System.out.printf(format, item1, ..., itemk)
```

where

- **format** is a string that may consist of substrings and *format specifiers.* A format specifier specifies how an item should be displayed. Each specifier begins with a percent sign.
- **item** may be a any primitive value or a string.

Format Specifier	*Output*	*Example*
%b	a Boolean value	true or false
%c	a character	'a'
%d	a decimal integer	200
%f	a floating-point number	45.460000
%s	a string	"Java is cool"

Formatting Console Output

You can specify the width and precision in a format specifier, as shown in the examples:

%5c Output the character and add four spaces **before** the character item, because the width is 5.

%6b Output the Boolean value and add one space before the false value and two spaces before the true value.

%5d Output the integer item with width at least 5.

%9.2f Output the floating-point item with width at least 9 including a decimal point and two digits after the point. Thus, there are 6 digits allocated before the decimal point.

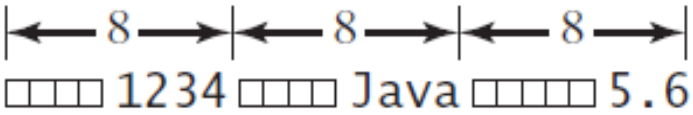
%8s Output the string with width at least 8 characters.

Notes:

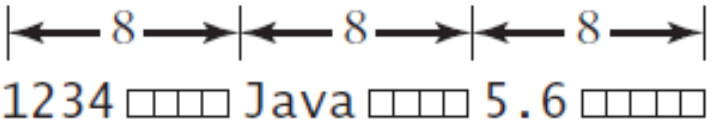
- If an item requires more spaces than the specified width, the width is automatically increased.
- By default, the output is right justified. You can put the minus sign (**-**) in the format specifier to specify that the item is left justified
- The % sign denotes a format specifier. To output a literal % in the format string, use %%.

Formatting Console Output: Example

System.out.printf(**"%8d%8s%8.1f\n"**, **1234**, **"Java"**, **5.63**);



System.out.printf(**"%-8d%-8s%-8.1f \n"**, **1234**, **"Java"**, **5.63**);



Reading Input from the Keyboard

Two steps to read input from the keyboard:

Step1) Create a Scanner object

```
import java.util.Scanner
…
Scanner input = new Scanner(System.in);
```

Step 2) Use an appropriate method (e.g., nextDouble()) to obtain a double value.

```
System.out.print("Enter a double value: ");
double d = input.nextDouble();
```

Example: Reading Input from the Keyboard

```
import java.util.Scanner; // Scanner is in the java.util package

public class ComputeAreaWithConsoleInput {
    /* Main method */
    public static void main(String[] args) {
        // Create a Scanner object
        Scanner input = new Scanner(System.in);

        // Prompt the user to enter a radius
        System.out.print("Enter a number for radius: ");
        double radius = input.nextDouble();

        // Compute area
        double area = radius * radius * 3.14159;

        // Display results

        System.out.println("The area for the circle of radius "
                           + radius + " is " + area);
    }
}
```

Step 1

Example: Reading Input from the Keyboard

```
import java.util.Scanner; // Scanner is in the java.util package

public class ComputeAreaWithConsoleInput {
    /* Main method */
    public static void main(String[] args) {
        // Create a Scanner object
        Scanner input = new Scanner(System.in);

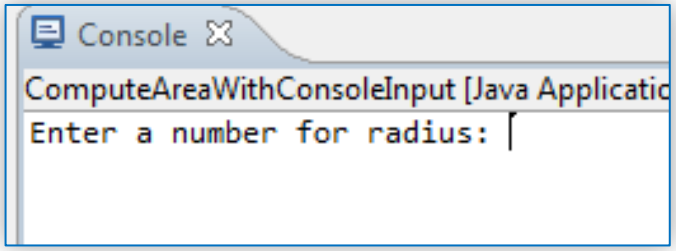
        // Prompt the user to enter a radius
        System.out.print("Enter a number for radius: ");
        double radius = input.nextDouble();

        // Compute area
        double area = radius * radius * 3.14159;

        // Display results

        System.out.println("The area for the circle of radius "
                           + radius + " is " + area);
    }
}
```

prompt the user for input.



Example: Reading Input from the Keyboard

```
import java.util.Scanner; // Scanner is in the java.util package

public class ComputeAreaWithConsoleInput {
    /* Main method */
    public static void main(String[] args) {
        // Create a Scanner object
        Scanner input = new Scanner(System.in);

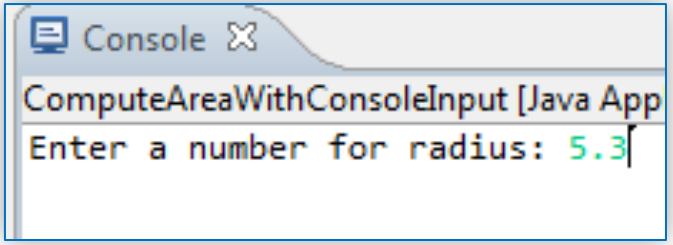
        // Prompt the user to enter a radius
        System.out.print("Enter a number for radius: ");
        double radius = input.nextDouble();

        // Compute area
        double area = radius * radius * 3.14159;

        // Display results

        System.out.println("The area for the circle of radius "
                           + radius + " is " + area);
    }
}
```

Step 2



Example: Reading Input from the Keyboard

```
import java.util.Scanner; // Scanner is in the java.util package

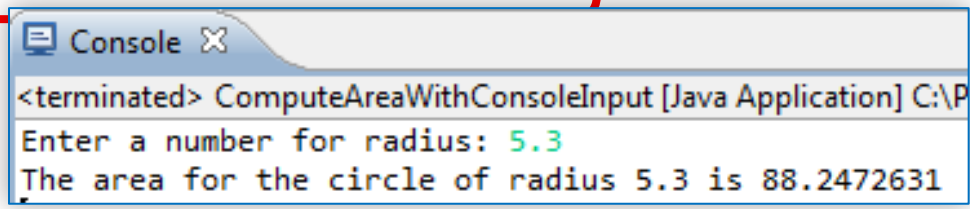
public class ComputeAreaWithConsoleInput {
    /* Main method */
    public static void main(String[] args) {
        // Create a Scanner object
        Scanner input = new Scanner(System.in);

        // Prompt the user to enter a radius
        System.out.print("Enter a number for radius: ");
        double radius = input.nextDouble();

        // Compute area
        double area = radius * radius * 3.14159;

        // Display results

        System.out.println("The area for the circle of radius "
                           + radius + " is " + area);
    }
}
```



Scanner Functions

Function	Description
`nextByte()`	reads an integer of the `byte` type.
`nextShort()`	reads an integer of the `short` type.
`nextInt()`	reads an integer of the `int` type.
`nextLong()`	reads an integer of the `long` type.
`nextFloat()`	reads a number of the `float` type.
`nextDouble()`	reads a number of the `double` type.
`next()`	reads a 'token' of the `String` type.
`nextLine()`	reads a line of text of the `String` type.

Redundant Input Objects

This code is not wrong, BUT inefficient!

```
Scanner input1 = new Scanner(System.in);
System.out.print("Enter an integer: ");
int v1 = input1.nextInt();

Scanner input2 = new Scanner(System.in);
System.out.print("Enter a double value: ");
double v2 = input2.nextDouble();
```

You should rewrite the above code as follows

```
Scanner input = new Scanner(System.in);
System.out.print("Enter an integer: ");
int v1 = input.nextInt();
System.out.print("Enter a double value: ");
double v2 = input.nextDouble();
```

Named Constants

A constant is declared like a variable but it must be declared and initialized in the same statement.

We use the keyword **`final`** to declare constants.

```
public class ComputeAreaWithConstant {
    public static void main(String[] args) {
        final double PI = 3.14159; // Declare a constant

        // Create a Scanner object
        Scanner input = new Scanner(System.in);

        // Prompt the user to enter a radius
        System.out.print("Enter a number for radius: ");
        double radius = input.nextDouble();

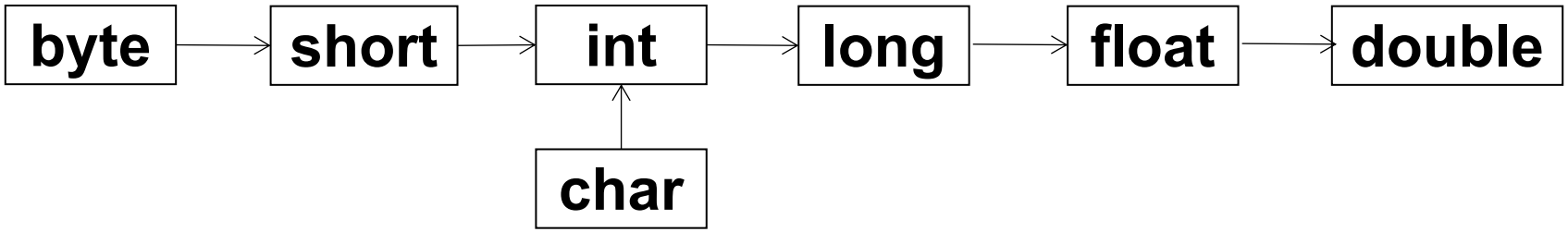
        // Compute area
        double area = radius * radius * PI;

        // Display result
        System.out.println("The area for the circle of radius " + radius
                + " is " + area);
    }
}
```

Numeric Type Conversion

Implicit Casting: You can always assign a value to a numeric variable whose type supports a larger range of values.

- `double x = 5; // no error (type widening)`



Explicit Casting: You cannot assign a value to a variable of a type with a smaller range unless you use ***type casting***.

- `int y = 3.5; // compilation error`
- `int z = (int) 3.5; // z = 3(type narrowing)`
- ```
  double d = 7.61;
  int n = (int) d; // n = 7. no change to d
  ```
  ```

Conversion Rules

When performing a binary operation involving two operands of different numeric types, Java automatically converts the operand based on the following rules:

- If one of the operands is double, the other is converted into double.
- Otherwise, if one of the operands is float, the other is converted into float.
- Otherwise, if one of the operands is long, the other is converted into long.
- Otherwise, both operands are converted into int.

The `Math` Class

Mathematical Functions and Constants

Java provides many useful methods in the **`Math`** class for performing common mathematical functions.

- *trigonometric methods*,
- *exponent methods*, and
- *service methods*

Two useful double constants,

- PI
- E (the base of natural logarithms).

Trigonometric Methods

```
Math.sin(r)
Math.cos(r)
Math.tan(r)
Math.acos(r)
Math.asin(r)
Math.atan(r)

Math.toRadians(d)
Math.toDegree(r)
```

Examples:

Math.toDegrees(Math.PI/2) returns 90.0

Math.toRadians(30) returns 0.5236 (i.e., π/6)

Math.sin(0) returns 0.0

Math.sin(Math.PI/6) returns 0.5

Math.sin(Math.toRadians(90)) returns 1.0

Math.cos(0) returns 1.0

Math.cos(Math.PI / 6) returns 0.866

Math.cos(Math.PI / 2) returns 0

Exponent Methods

`Math.pow(a, b)`

- returns a raised to power of b.

`Math.sqrt(a)`

- returns square root of a.

`Math.exp(a)`

- returns e raised to power of a.

`Math.log(a)`

- returns natural logarithm of a.

`Math.log10(a)`

- returns the 10-based logarithm of a.

Examples:

Math.exp(1) returns 2.71

Math.log(2.71) returns 1.0

Math.pow(2, 3) returns 8.0

Math.pow(3, 2) returns 9.0

Math.pow(3.5, 2.5) returns 22.9176

Math.sqrt(4) returns 2.0

Math.sqrt(10.5) returns 3.24

Rounding Methods

`Math.ceil(x)`

- x rounded up to its nearest integer.

`Math.floor(x)`

- x is rounded down to its nearest integer.

`Math.rint(x)`

- x is rounded to its nearest integer.
- If x is equally close to two integers, the ***even*** one is returned

`Math.round(x)`

- Return (int)Math.floor(x+0.5).

In all methods, the result is returned as a double value.

Rounding Methods Examples

Examples:

Math.ceil(2.1) returns 3.0

Math.ceil(2.0) returns 2.0

Math.ceil(-2.0) returns –2.0

Math.ceil(-2.1) returns -2.0

Math.floor(2.1) returns 2.0

Math.floor(2.0) returns 2.0

Math.floor(-2.0) returns –2.0

Math.floor(-2.1) returns -3.0

Math.rint(2.1) returns 2.0

Math.rint(2.0) returns 2.0

Math.rint(-2.0) returns –2.0

Math.rint(-2.1) returns -2.0

Math.rint(2.5) returns 2.0

Math.rint(-2.5) returns -2.0

Math.round(2.6f) returns 3

Math.round(2.0) returns 2

Math.round(-2.0f) returns -2

Math.round(-2.6) returns -3

min, max, and abs methods

`Math.max(a, b)`

`Math.min(a, b)`

- Return the maximum or minimum of a and b.

`Math.abs(a)`

- Returns the absolute value of a.

Examples:

Math.max(2, 3) returns 3

Math.max(2.5, 3) returns 3.0

Math.min(2.5, 3.6) returns 2.5

Math.abs(-2) returns 2

Math.abs(-2.1) returns 2.1

The random Method

`random()`

- Returns a random double value in the range [0.0, 1.0).
  - 0 **<=** Math.random() **<** 1.0

Examples:

```
(int)(Math.random() * 10)
```
→ Returns a random integer between 0 and 9.

```
50 + (int)(Math.random() * 50)
```
→ Returns a random integer between 50 and 99.

In general,

```
a + Math.random() * b
```
→ Returns a random number between a and a + b, excluding a + b.

The `Character` Class

Character Data Type

The character data type, **char**, is used to represent a single character.

A character literal is enclosed in single quotation marks.

Examples:

```
char letter = 'A';
char numChar = '4';
```


```
char letter = '\u0041';
char numChar = '\u0034';
```

The ++ and -- operators can be used on char variables to get the next or preceding Unicode character. For example.

```
char ch = 'a';
System.out.println(++ch); // displays character b
```

Casting between char and Numeric Types

A char can be cast into numeric types, and vice versa.

```
int i = 'A';         // decimal value of A which is 65 is stored in I
int i = (int) 'A'; // Same as above
char c = 97;    // Same as char c = (char)97;
```

floating-point values (must be explicit)

```
char ch = (char)65.25;  // Decimal 65 is assigned to ch
```

Comparing and Testing Characters

Characters can be compared based on their Unicode values.

Examples:

'1' < '8'

- **True** because the Unicode for **'1'** (**49**) is less than the Unicode for **'8'** (**56**).

'a' < 'b'

- **True** because the Unicode for **'a'** (**97**) is less than the Unicode for **'b'** (**98**).

'a' < 'A'

- **False** because the Unicode for **'a'** (**97**) is greater than the Unicode for **'A'** (**65**).

Methods in the `Character` Class

`isDigit(ch)`

- Returns true if the specified character is a digit.

`isLetter(ch)`

- Returns true if the specified character is a letter.

`isLetterOrDigit(ch)`

- Returns true if the specified character is a letter or digit.

`isLowerCase(ch)`

- Returns true if the specified character is a lowercase letter.

`isUpperCase(ch)`

- Returns true if the specified character is an uppercase letter.

`toLowerCase(ch)`

- Returns the lowercase of the specified character.

`toUpperCase(ch)`

- Returns the uppercase of the specified character.

Methods in the `Character` Class

For example,

- Character.isDigit('a') returns false
- Character.isLetter('a') returns true
- Character.isLowerCase('a') returns true
- Character.isUpperCase('a') returns false
- Character.toLowerCase('T') returns t
- Character.toUpperCase('q') returns Q

The `String` Type

The String Type

To represent *a sequence of characters*, use the data type called **String**.

```
String message = "Welcome to Java";
```

The String type is **not a primitive type**.

- String is actually a predefined class in the Java library just like the System class and Scanner class. It is known as a reference type.
  - Any Java class can be used as a reference type for a variable. Reference data types will be thoroughly discussed in Chapter 9,
  - For the time being, you just need to know how to declare a String variable, how to assign a string to the variable, how to concatenate strings, and to perform simple operations for strings.

Methods for String Objects

`s1.length()`

- Returns the number of characters in the string s1.

  "Welcome".length() returns 7

`s1.charAt(index)`

- Returns the character at the specified index from string s1.

  "Welcome".charAt(0) returns 'W'

`String s2 = s1.toUpperCase()`

- Returns a new string s2 with all letters of s1 in uppercase.

  "Welcome".toUpperCase() returns a new string, WELCOME

`String s2 = s1.toLowerCase()`

- Returns a new string with all letters in lowercase.

  "Welcome".toLowerCase() returns a new string, welcome

`s1.trim()`

- Trims whitespace characters on both sides of s1.

  " Welcome ".trim() returns a new string, Welcome

Methods for String Objects

Strings are objects in Java.

The methods in the preceding table can only be invoked from a **specific string instance**. For this reason, these methods are called instance methods.

- e.g,

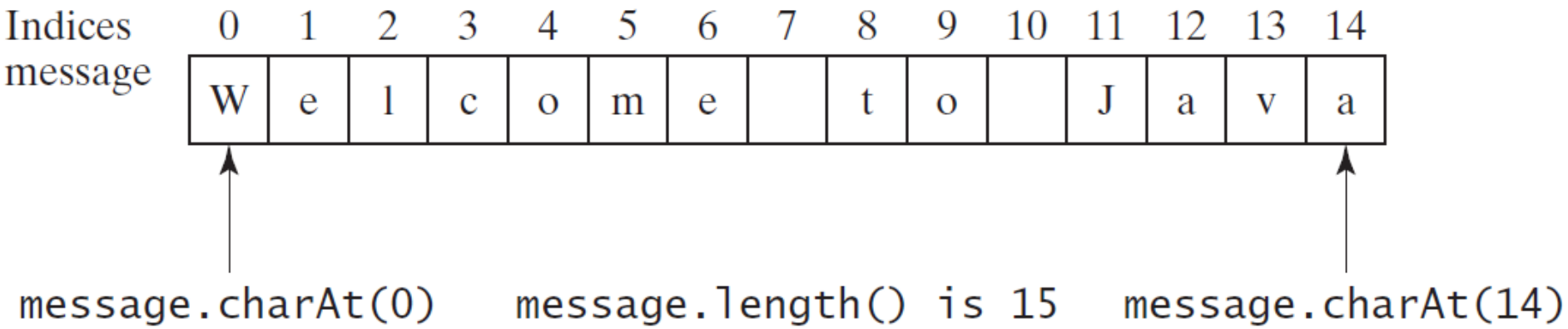
```
String s = "abc";
int x = s.length()
```

A non-instance method is called a static method. A static method can be invoked without using an object.

- All the methods defined in the **Math class are static methods**. They are not tied to a specific object instance. They can be invoked directly using the Math class. e.g.,

```
Math.sin(Math.PI/2)
```

String: charAt()




```
String s= "Welcome to Java";
System.out.println("First character is "
                   + s.charAt(0));
```

Reading a String from the Console

You can use a Scanner object to read a string from the console.

You may use the methods:

- next().
  - To reads a ‘token’.
- nextLine().
  - To read a line of text (ends with *newline* character)
  - The newline character is not read.

Reading a String from the Console

Using next() method.

```
import java.util.Scanner;
public class Ex1 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter 2 words separated by spaces: ");
        String s1 = input.next();
        String s2 = input.next();
        System.out.println("s1 is " + s1);
        System.out.println("s2 is " + s2);
    }
}
```

Using nextLine() method

```
import java.util.Scanner;
public class Ex1 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter a line: ");
        String s = input.nextLine();
        System.out.println("The line entered is " + s);
    }
}
```

More methods: Comparing Strings

`s1.equals(s2)`

- returns ***true*** if s1 is equal to s2

`s1.equalsIgnoreCase(s2)`

- same as equals but it is case insensitive.

`s1.compareTo(s2)`

- returns an ***integer*** > 0, = 0, or < 0 to indicate whether s1 is greater than, equal to, or less than s2.

`s1.compareToIgnoreCase(s2)`

- same as compareTo except that it is case insensitive

`s1.startsWith(prefix)`

- returns ***true*** if s1 starts with the specified prefix.

`s1.endsWith(suffix)`

- Returns true if s1 ends with the specified suffix.

compareTo()

The method returns

- **0** if **s1** is equal to **s2**
- **Negative value** if **s1** is lexicographically less than **s2**, and
- **Positive value** if **s1** is lexicographically greater than **s2**.

The value returned from the **compareTo** method depends on the **offset of the first two different characters** in **s1** and **s2** from left to right.

Example:

Assume s1 is "**abc**" and **s2** is "**abe**"

s1.compareTo(s2) returns **-2**.

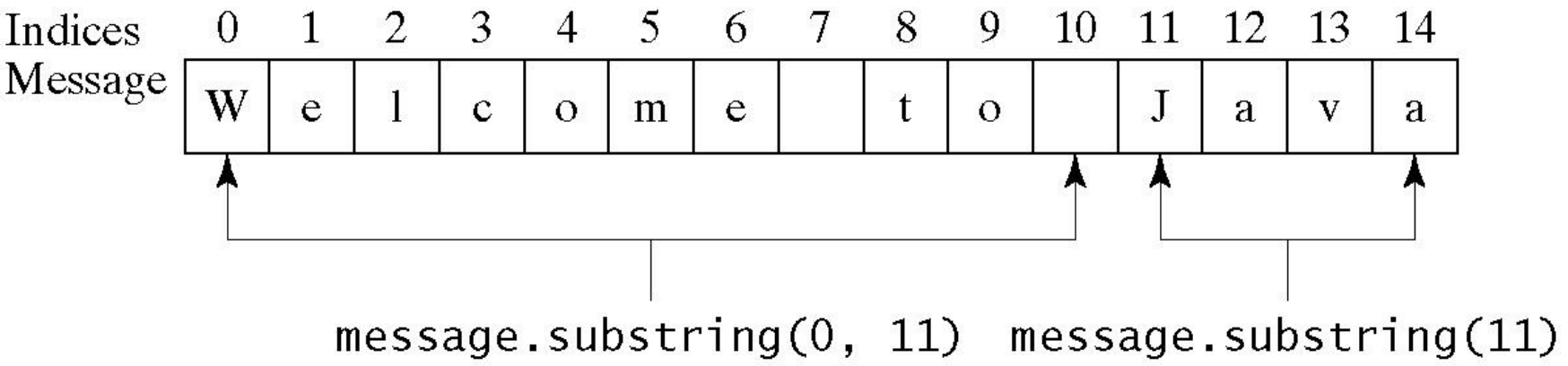
substring()

`substring(beginIndex)`

- Returns this string's substring that begins with the character at the specified beginIndex and extends to the end of the string.

`substring(beginIndex, endIndex)`

- Returns this string's substring that begins at the specified beginIndex and extends to the character at index endIndex – 1,. Note that the character at endIndex **is not part of** the substring.



indexOf()and lastIndexOf()

s1.indexOf(s)

- Returns index of the first occurrence of s in the s1.

s1.indexOf(s, fromIndex)

- Returns index of the first occurrence of s after fromIndex in s1.

s1.lastIndexOf(s)

- Returns index of the last occurrence of s in s1.

s1.lastIndexOf(s, fromIndex)

- Returns index of last occurrence of s before fromIndex in s1

All above methods

- **return -1 if no match is found.**
- **s could be a character or a string**

Example1: Finding a Character/Substring

"Welcome to Java".indexOf('W') returns 0.
"Welcome to Java".indexOf('o') returns 4.
"Welcome to Java".indexOf('o', 5) returns 9.
"Welcome to Java".indexOf("come") returns 3.
"Welcome to Java".indexOf("Java", 5) returns 11.
"Welcome to Java".indexOf("java", 5) returns -1.

"Welcome to Java".lastIndexOf('W') returns 0.
"Welcome to Java".lastIndexOf('o') returns 9.
"Welcome to Java".lastIndexOf('o', 5) returns 4.
"Welcome to Java".lastIndexOf("come") returns 3.
"Welcome to Java".lastIndexOf("Java", 5) returns -1.
"Welcome to Java".lastIndexOf("Java") returns 11.

Example2: Extracting two words from a string

```
int k = s.indexOf(' ');
String firstName = s.substring(0, k);
String lastName = s.substring(k + 1);
```