CMPS 200: Introduction to Programming Using JAVA

LECTURE 6 – Output Formatting and Wrapper Classes

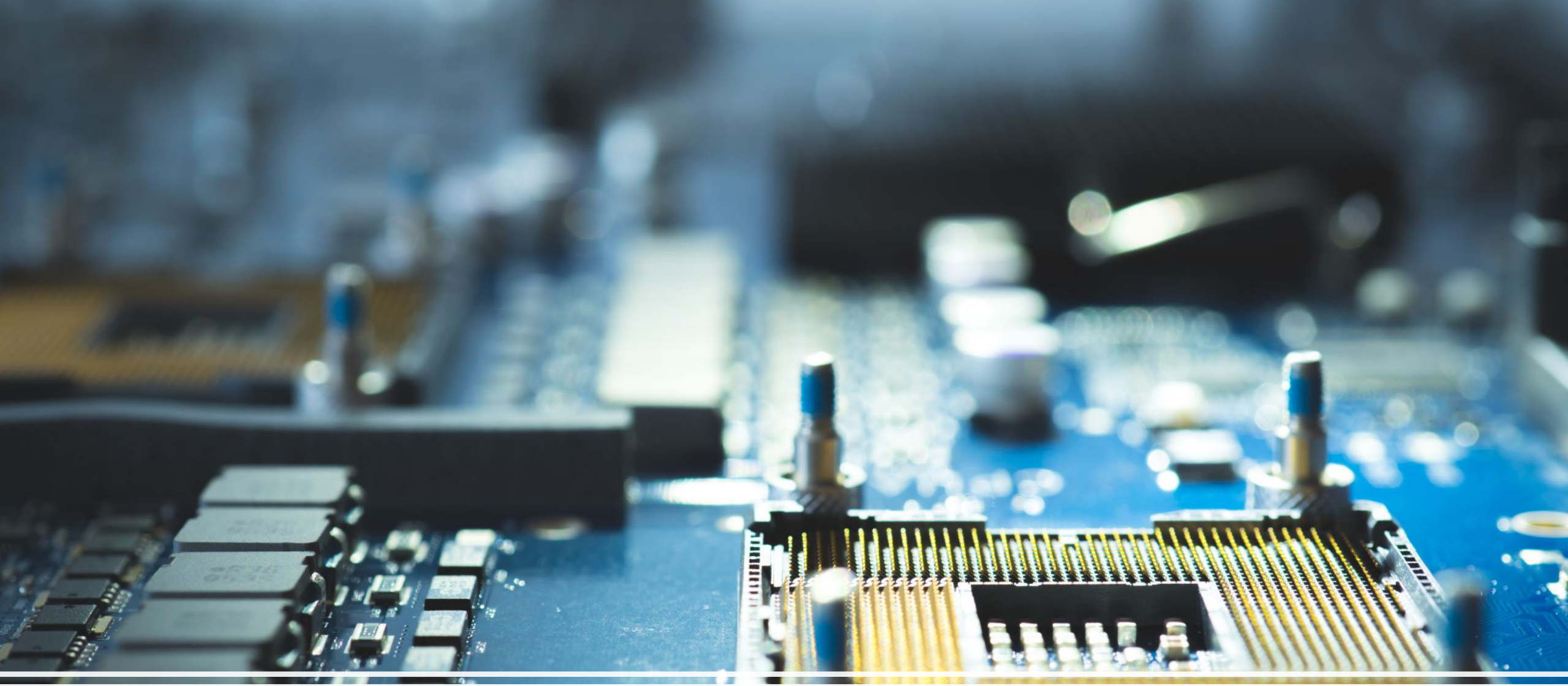
Maurice J. KHABBAZ, Ph.D.

Last Lecture

- **Algorithms:**
  1. Complex loops
  2. Guess and Check
  3. Approximate Solutions
  4. Searching:
     - Linear Search
     - Binary Search

In This Lecture

1. Output Formatting:
   - `NumberFormat` Class.
   - `DecimalFormat` Class.
   - `printf()` method.
2. Wrapper Classes:
   - `Integer` Class
   - `Double` Class
   - `Character` Class



Output Formatting

Two Main Output Formatting Classes

- Include methods to format information to look appropriate when displayed.
- Both classes are part of the JAVA Standard Library's `java.text` package.

- **`NumberFormat`** class.

- **`DecimalFormat`** class.

`NumberFormat` Class

- Provides generic formatting capabilities for numbers.
- No object instantiation required.
- Request objects from static methods invoked using class name itself.
- **Available Methods:**

Method	Description
`format(double n)`	Returns a `String` containing the specified number n formatted to this object's pattern.
`getCurrencyInstance()`	Returns a `NumberFormat` object that represents a currency format for the current locale.
`gerPercentInstance()`	Returns a `NumberFormat` object that represents a percentage format for the current locale.

Example: NumberFormat Class

```
import java.util.Scanner; import java.text.NumberFormat;
public class Purchase {
   public static void main(String[] args) {
      final double TAX_RATE = 0.06;  // 6% sales tax.
      int quantity; double subtotal, tax, totalCost, unitPrice;
      Scanner keyb = new Scanner(System.in);
      NumberFormat fmtC = NumberFormat.getCurrencyInstance();
      NumberFormat fmtP = NumberFormat.getPercentInstance();
      System.out.print("Quantity: "); quantity = keyb.nextInt();
      System.out.print("Unit Price: "); unitPrice = keyb.nextDouble();
      subtotal = quantity * unitPrice; tax = subtotal * TAX_RATE;
      totalCost = subtotal + tax;
      System.out.println("Subtotal: " + fmtC.format(subtotal));
      System.out.println("Tax: " + fmtC.format(tax) + " at "
                                + fmtP.format(TAX_RATE) + " rate");
      System.out.println("Total: " + fmtC.format(totalCost));
   }
}
```

Sample Execution:

```
Quantity: 5
Unit Price: 3.87
Subtotal: $19.35
Tax: $1.16 at 6% rate
Total: $20.51
```

`DecimalFormat` Class

- Requires object instantiation using the `new` operator.
- Constructor method takes a `String` as parameter:
  - Represents the pattern that will guide the formatting process using various symbols.
- **Example:**
  - `"0.###"` → at least one digit to the left of decimal point and fractional part rounded to 3 digits after the decimal point.

- Use the `format()` method to format a particular value.
- To change the formatting pattern, use the `applyPattern()` method.

Example: DecimalFormat Class

```
import java.util.Scanner; import java.text.DecimalFormat;
public class CircleStats {
    public static void main(String[] args) {
        double r, a, p; Scanner k = new Scanner(System.in);
        DecimalFormat fmt = new DecimalFormat("0.###");
        System.out.print("Radius: "); r = k.nextDouble();
        a = Math.PI * Math.pow(r,2); p = 2 * Math.PI * r;
        System.out.println("Area: " + fmt.format(a));
        System.out.println("Perimeter: " + fmt.format(p));
    }
}
```

Sample Execution:

```
Radius: 5
Area: 78.54
Perimeter: 31.416
```

Formatting Text Using The `printf()` Method

- **Syntax:** `System.out.printf("<format_str>", <params>);`
- `<format_str>` contains placeholders to insert parameters `<params>`:
  - Placeholders used **instead of + concatenation**.
  - **`%d`** → indicates the insertion of an **`int`**.
  - **`%f`** → indicates the insertion of an **`double or float`**.
  - **`%s`** → indicates the insertion of an **`String`**.
- **Example 1:**

```
int x = 3; double y = -17.6;
System.out.printf("x = %d and y = %f\n", x, y);
```

- **Remark:**
  - `printf()` doesn't go to a new line unless escape character `\n` is included in `<format_str>`.

`printf()` Method: Additional Placeholders

Placehoder	Description
`%Wd`	int, `W` characters (chars.) wide, right-alligned
`%-Wd`	int, `W` chars. wide, left-alligned
`%Wf`	`double` / `float`, `W` chars. wide, right-alligned
`%-Wf`	`double` / `float`, `W` chars. wide, left-alligned
`%.Df`	`double` / `float`, rounded to `D` digits after decimal (dec.) point
`%W.Df`	`double` / `float`, `W` chars. wide, `D` digits after dec. point, right-alligned
`%-W.Df`	`double` / `float`, `W` chars. wide, `D` digits after dec. point, left-alligned

Example 2: `printf()` Method

What output is generated following the execution of the below JAVA code snippet?

```
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 10; j++) {
        System.out.printf("%4d", (i * j));
    }
    System.out.println();
}
```

OUTPUT

```
   1   2   3   4   5   6   7   8   9  10
   2   4   6   8  10  12  14  16  18  20
   3   6   9  12  15  18  21  24  27  30
```

Example 2: `printf()` Method

What output is generated following the execution of the below JAVA code snippet?

```
double gpa = 3.253764;
System.out.printf("Your GPA is %.1f\n", gpa);
System.out.printf("More precisely: %8.3f\n", gpa);
```

OUTPUT

```
your GPA is 3.3
more precisely:    3.254
```

3

8

Wrapper Classes

Wrapper Classes

- In addition to classes and objects, JAVA represents data using primitive types:
  - `int`, `long`, `double`, `float`, `char` and `boolean`.

- **Objects:**
  - Serve as containers to hold various types of other objects.
  - Have a set of methods that describe their actions and can be called on them.

- Sometimes, interest lies in encapsulating one primitive type into an object:
  - **Example:** have an object contain a single `int` value.
- Need to **wrap** that primitive value into an object.

Wrapper Classes

- Represent respective primitive data types.
- Define methods used to manage values of these primitive types:
  - Some of these methods are static (invoked regardless of instantiated object).
- For each primitive type in JAVA there exists a wrapper class:
  - These classes are part of the `java.lang` package.

Primitive type	Wrapper class
`byte`	`Byte`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`float`	`Float`
`double`	`Double`
`char`	`Character`
`boolean`	`Boolean`

`Character` Class

- **Recall:**
  - `char` is a primitive data type representing **single** characters.
    - **Example:** `char letter = 'P';` ← single appostrophes
  - A `String` comprises multiple characters and internally stored as an **array** of `char`:
    - Each character in the `String` is numerically indexed by a positive integer (*i.e.* `>= 0`).
  - Characters in a `String` are referenced using the `charAt()` method.
  - A `for` loop is used to examine / print each character in a `String`.
  - Single characters may be compared using relational operators (!=, ==, <, >, >=, <=)
  - Each `char` is internally mapped to an `int` referred to as an ASCII value:
    - **Example:** `'A' → 65` , `'b' → 98`, …
  - Mixing `char` and int causes automatic conversion to `int`:
    - **Example:** `'A' + 10 → 75` , `'b' + 2 → 100`
  - To convert an `int` into its equivalent `char` → typecast it.
    - **Example:** `(char) 65 → 'A'` , `(char) ('a' + 2) → 'c'`
  - Because `char` is a primitive type → no methods can be called on it:
    - **Example:** `char c = 'a'; c.toUpperCase()` → **ERROR**

`Character` Class

`boolean isDigit(char)` : Checks if `char` is `'0'` through `'9'`

- **Example:** `Character.isDigit('X')` returns `false`

`boolean isLetter(char)` : Checks if `char` is in range `'a'` to 'z' or `'A'` to `'Z'`

- **Example:** `Character.isLetter('f')` returns `true`

`boolean isLowerCase(char)` : Checks if `char` is a lowercase letter

- **Example:** `Character.isLowerCase('Q')` returns `false`

`boolean isUpperCase(char)` : Checks if `char` is an uppercase letter

- **Example:** `Character.isUpperCase('Q')` returns `true`

`char toLowerCase(char)` : Returns the lowercase version of the given letter

- **Example:** `Character.toLowerCase('Q')` returns `'q'`

`char toUpperCase(char)` : Returns the uppercase version of the given letter

- **Example:** `Character.toUpperCase('x')` returns `'X'`

`char getNumericValue(char)` : Returns the numeric equivalent of a given `char` digit

- **Example:** `Character.getNumericValue('3')` returns `3 // as int value`

`Integer` Class

Method	Description
`Integer(int i)`	Constructor method: creates a new `Integer` object storing `i`
`Integer(String s)`	Overridden Constructor method: creates a new `Integer` object storing the integer representation of `s`.
`floatValue()`	Returns a floating-point version of the boxed / wrapped integer `i`
`parseInt(String s)`	A static method that returns an `int` corresponding to the value in `s`
`toString()`	Return the `String` version of the boxed / wrapped integer `i`

`byteValue()`
`doubleValue()`
`intValue()`
`longValue()`
	Return the value of the boxed / wrapped integer `i` as the corresponding primitive type

`toBinaryString(int n)`
`toHexString(int n)`
`toOctalString(int n)`
	Static methods that return a `String` representation of the specified integer parameter `n` in the corresponding base numbering system

Example: String to Integer (S2I) Conversion

Write a JAVA program that takes from the user a String called `i_str` representing an integer. The program will generate the actual `int` value `i_str` that will be stored in a variable called `i`, which can be used later in mathematical operations.

```
public class S2I {
    public static void main (String[] args) {
        Scanner k = new Scanner(System.in);
        String i_str; int i = 0;
        System.out.print("Enter an int: "); i_str = k.next();

        int l = i_str.length();
        for (int c = 0; c < l; c++)
            i += (i_str.charAt(c) - '0') * Math.pow(10,l - (c + 1));

        System.out.println("String version of i: " + i_str);
        System.out.println("Integer version of i = " + i);
        System.out.println("Verification: i + 2 = " + (i + 2));
    }
}
```

Example: S2I Conversion Using Character Class

Write a JAVA program that takes from the user a String called `i_str` representing an integer. The program will generate the actual `int` value `i_str` that will be stored in a variable called `i`, which can be used later in mathematical operations.

```
public class S2I {
    public static void main (String[] args) {
        Scanner k = new Scanner(System.in);
        String i_str; int i = 0;
        System.out.print("Enter an int: "); i_str = k.next();

        int l = i_str.length();
        for (int c = 0; c < l; c++)
            i += Character.getNumericValue(i_str.charAt(c)) *
                                        Math.pow(10,l - (c + 1));;

        System.out.println("String version of i: " + i_str);
        System.out.println("Integer version of i = " + i);
        System.out.println("Verification: i + 2 = " + (i + 2));
    }
}
```

Example: S2I Conversion using `Integer` Class

Write a JAVA program that takes from the user a String called `i_str` representing an integer. The program will generate the actual `int` value `i_str` that will be stored in a variable called `i`, which can be used later in mathematical operations.

```
public class S2I {
    public static void main (String[] args) {
        Scanner k = new Scanner(System.in);
        String i_str;
        System.out.print("Enter an int: "); i_str = k.next();

         int i = Integer.parseInt(i_str);

        System.out.println("String version of i: " + i_str);
        System.out.println("Integer version of i = " + i);
        System.out.println("Verification: i + 2 = " + (i + 2));
    }
}
```

`Double` Class

Method	Description
`Double(double d)`	Constructor method: creates a new `Double` object storing `d`
`Double(String s)`	Overridden Constructor method: creates a new `Double` object storing the floating-point (type `double`) representation of `s`.
`floatValue()`	Returns a floating-point version of the boxed / wrapped real number `d`
`parseDouble(String s)`	A static method that returns a `double` corresponding to the value in `s`
`toString()`	Return the `String` version of the boxed / wrapped floating-point `d`
`intValue()`	Returns the type-casted int value of the boxed / wrapped floating-point value `d`

Autoboxing and Unboxing

- **Autoboxing:**
  - Automatic conversion of a primitive value into its corresponding wrapper object.
- **Example:**

```
Integer obj;
int num = 69;
obj = num; // automatically creates an Integer object
```

- **Unboxing:**
  - Automatic extraction of a boxed primitive value from its corresponding wrapper object.
- **Example:**

```
Double obj = new Double (72.85);
double num;
num = obj; // automatically extracts the double value
```

Practice Exercise: S2D Conversion

Write a JAVA program that takes as input from the user a String called `d_str` composed of a floating-point number. Then, the program must generate the actual `double` value of `d_str` that will be stored in a variable called `d`, which can be used later in mathematical operations. The program must be implemented in two different versions:

- **Version 1:** Not using any wrapper class.
- **Version 2:** Using the `Double` class.

Java™