420-101-VA: Programming 1

WEEK 2: INTERACTIVE PROGRAMS

The `Scanner` Class

- Programs generally need input on which to operate
- The class `Scanner` contains the pre-defined methods for reading input values of various types
- The `Scanner` class is not automatically available to your Java programs. Any program that uses the `Scanner` class should have the following statement:

```
import java.util.Scanner;
```

- The `Scanner` class is defined in `java.util`, so we will use the `import` statement at the top of our programs.
- Notice, import should be before any class definition. (You can use auto-completion to import a class)

The `Scanner` Class

- Once we have imported the Scanner class, we can use it to input.
- Scanner objects work with `System.in`
- To create a `Scanner` object:

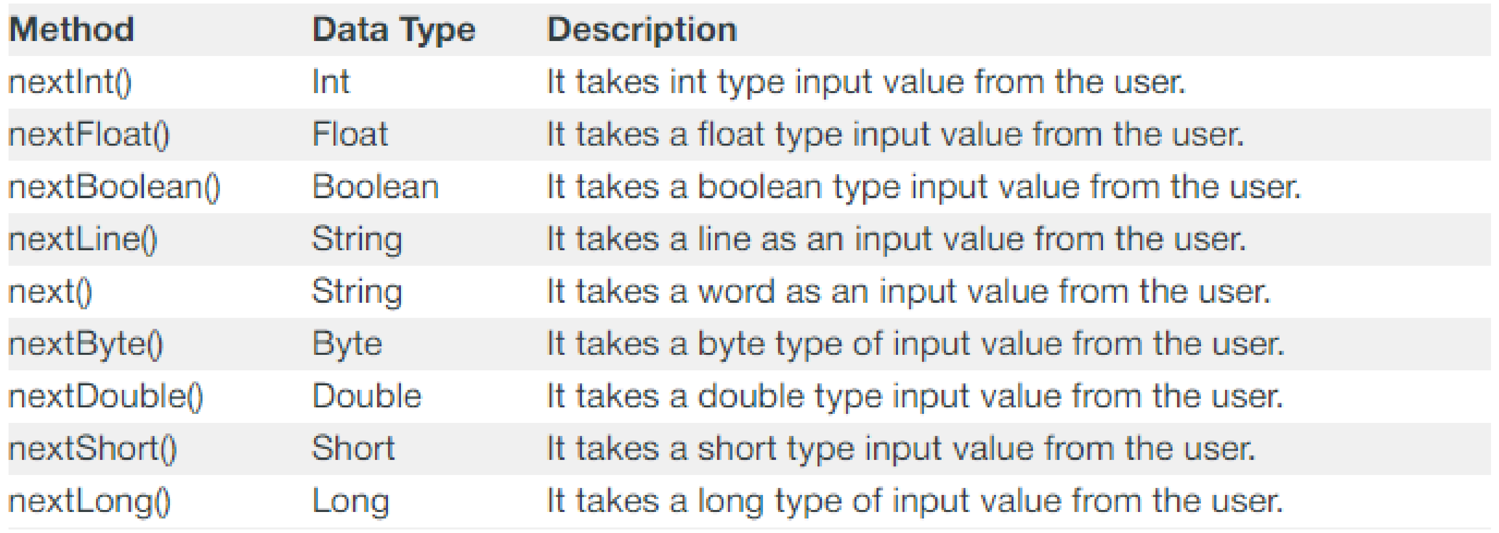
```
Scanner keyboard = new Scanner (System.in);
```

- This is a long statement, and right now you may not fully understand what it is doing. *Try to memories it*.

- And remember that `Scanner` is a class, `keyboard` is an object. Class looks like a data type here.

The `Scanner` Class

- The first two steps are only preparing for the user input.
- After we create an object of the Scanner class, we can write the real statement to ask the user to input.
- Depending on the **input datatype**, there are `Scanner` class methods for user to input



Method	Data Type	Description
nextInt()	Int	It takes int type input value from the user.
nextFloat()	Float	It takes a float type input value from the user.
nextBoolean()	Boolean	It takes a boolean type input value from the user.
nextLine()	String	It takes a line as an input value from the user.
next()	String	It takes a word as an input value from the user.
nextByte()	Byte	It takes a byte type of input value from the user.
nextDouble()	Double	It takes a double type input value from the user.
nextShort()	Short	It takes a short type input value from the user.
nextLong()	Long	It takes a long type of input value from the user.

Input Tokens

- Unless specified otherwise, *white space* is used to separate the elements (called *tokens*) of the input
- White space includes space characters, tabs, new line characters
- The `next` method of the `Scanner` class reads the next input token and returns it as a string
- Methods such as `nextInt` and `nextDouble` read data of particular types

```java
//Scanner Example
import java.util.Scanner;
public class Sum {
    public static void main(String[] args) {
        int num1;
        int num2;
        int sum;
        Scanner console = new Scanner(System.in);
        System.out.println("Please enter the value of num1");
        num1 = console.nextInt();
        System.out.println("Please enter the value of num2");
        num2 = console.nextInt();
        sum = num1 + num2;
        System.out.println(num1 + " + " + num2 + " = " + (num1 + num2));
    }
}
```

The **`String`** Class

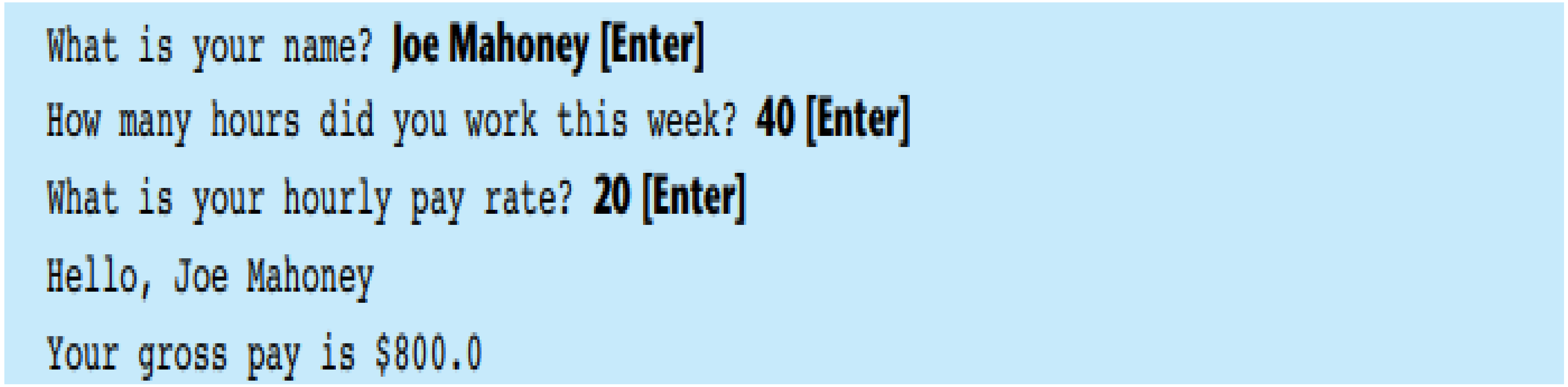
- Java has no primitive data type that holds a series of characters.
- The `String` class from the Java standard library is used for this purpose.
- A variable must be created to reference a `String` object.

```
String cityName = “Montreal";
```

- Notice the `S` in `String` is upper case.
- By convention, class names should always begin with an uppercase character.
- *We will visit String Class later on this course*

Demo..

- Payroll program shows the Scanner class being used to read a String, an int, and a double. The following output will be produced:



```
What is your name? Joe Mahoney [Enter]
How many hours did you work this week? 40 [Enter]
What is your hourly pay rate? 20 [Enter]
Hello, Joe Mahoney
Your gross pay is $800.0
```

```java
public static void main(String[] args) {
      String name;         // To hold a name
      int hours;           // Hours worked
      double payRate;      // Hourly pay rate
      double grossPay;     // Gross pay
      // Create a Scanner object to read input.
      Scanner input = new Scanner(System.in);
      // Get the user's name.
      System.out.print("What is your name? ");
      name = input.nextLine();
      // Get the number of hours worked this week.
      System.out.print("How many hours did you work this week? ");
      hours = input.nextInt();
      // Get the user's hourly pay rate.
      System.out.print("What is your hourly pay rate? ");
      payRate = input.nextDouble();
      // Calculate the gross pay.
      grossPay = hours * payRate;
      // Display the resulting information.
      System.out.println("Hello, " + name);
      System.out.println("Your gross pay is $" + grossPay);
   }
```

The `System.out.printf` Method

- You can use the `System.out.printf` method to perform formatted console output.
- The general format of the method is:

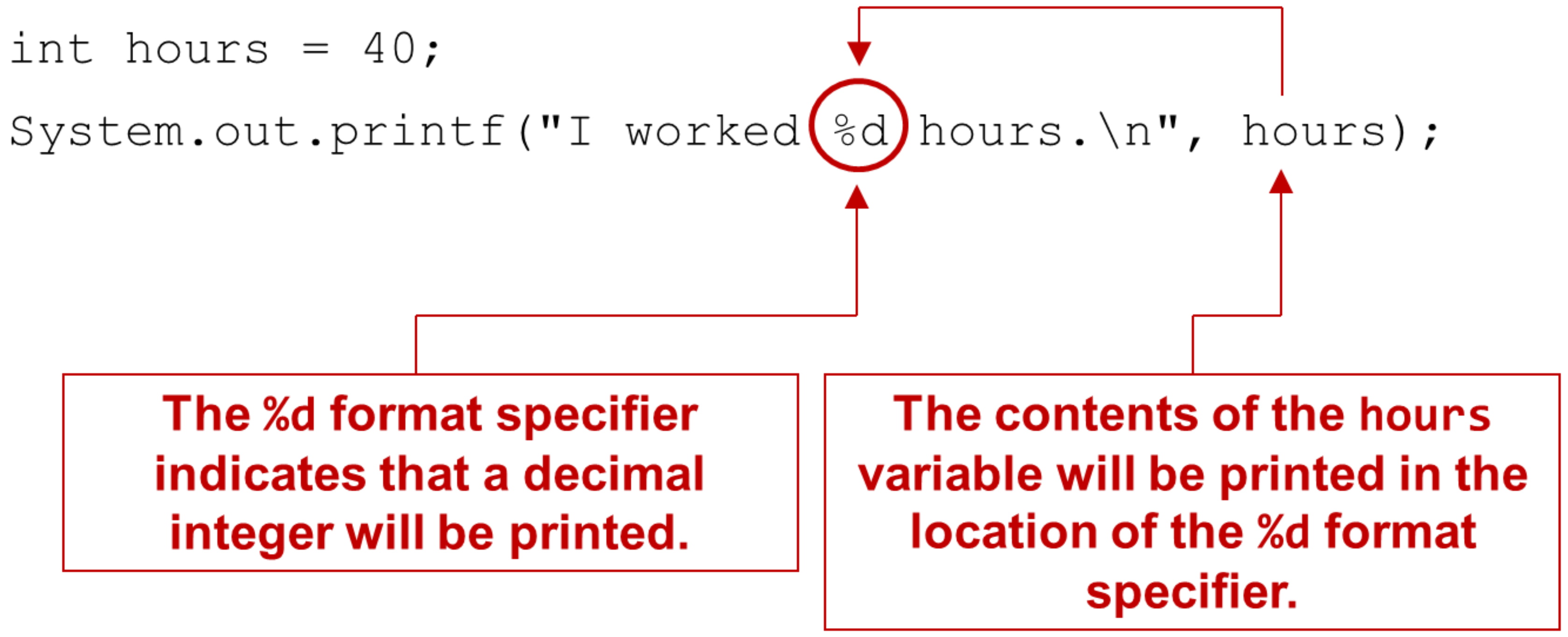
```
System.out.printf(FormatString, ArgList);
```

FormatString **is a string that contains text and/or special formatting specifiers.**

ArgList **is optional. It is a list of additional arguments that will be formatted according to the format specifiers listed in the format string.**

The `System.out.printf` Method

- A simple example:



```
int hours = 40;
System.out.printf("I worked %d hours.\n", hours);
```

The %d format specifier indicates that a decimal integer will be printed.

The contents of the hours variable will be printed in the location of the %d format specifier.

```
I worked 40 hours.
```

The `System.out.printf` Method

- Another example:

```
double grossPay = 874.12;
System.out.printf("Your pay is %f.\n", grossPay);
```

The `%f` format specifier indicates that a floating-point value will be printed.

The contents of the `grossPay` variable will be printed in the location of the `%f` format specifier.

```
Your pay is 874.12
```

The `System.out.printf` Method

- Another example:

```
double grossPay = 874.12;
System.out.printf("Your pay is %.2f.\n", grossPay);
```

The `%.2f` format specifier indicates that a floating-point value will be printed, rounded to two decimal places.

```
Your pay is 874.12
```

The `System.out.printf` Method

- Another example:

```
double grossPay = 5874.127;
System.out.printf("Your pay is %,.2f\n", grossPay);
```

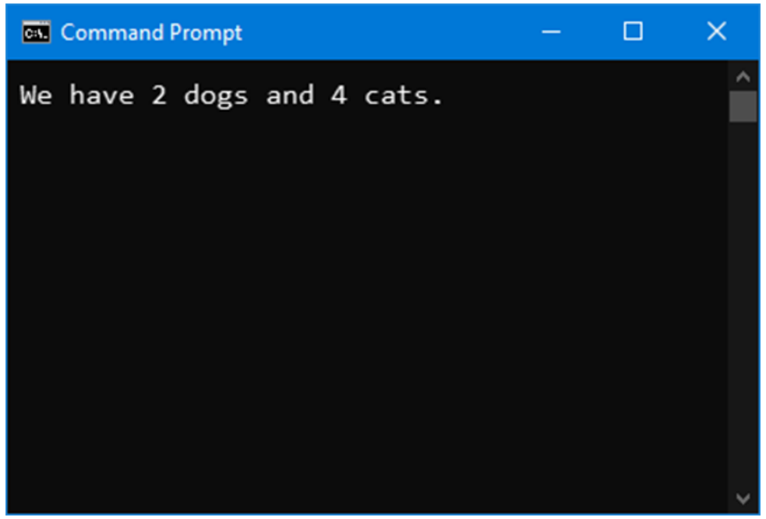
The `%,.2f` format specifier indicates that a floating-point value will be printed with comma separators, rounded to two decimal places.

```
Your pay is 5,874.13
```

The `System.out.printf` Method

- Another example:

```
int dogs = 2, cats = 4;
System.out.printf("We have %d dogs and %d cats.\n",
                  dogs, cats);
```



```
We have 2 dogs and 4 cats.
```

The `System.out.printf` Method

- Another example:

```
String name = "Ringo";
System.out.printf("Your name is %s.\n", name);
```

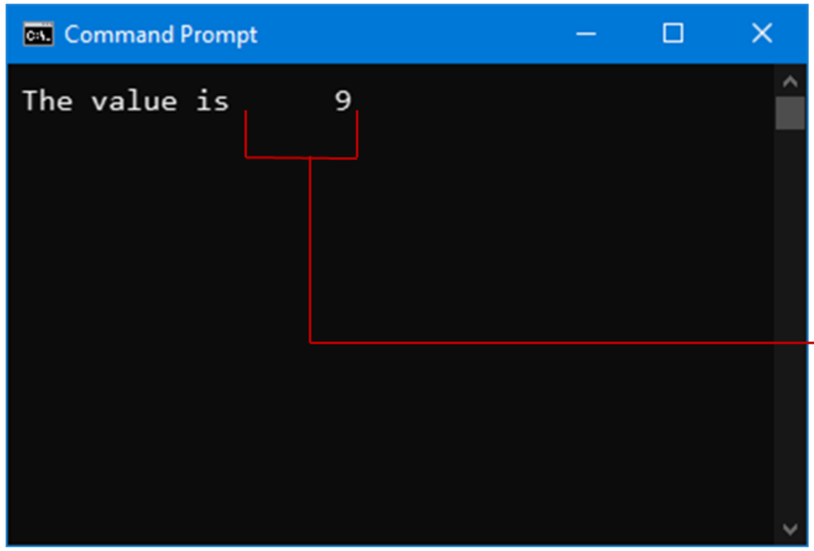
The `%s` format specifier indicates that a string will be printed.

```
Your name is Ringo.
```

The `System.out.printf` Method

- Specifying a field width:

```
int number = 9;
System.out.printf("The value is %6d\n", number);
```



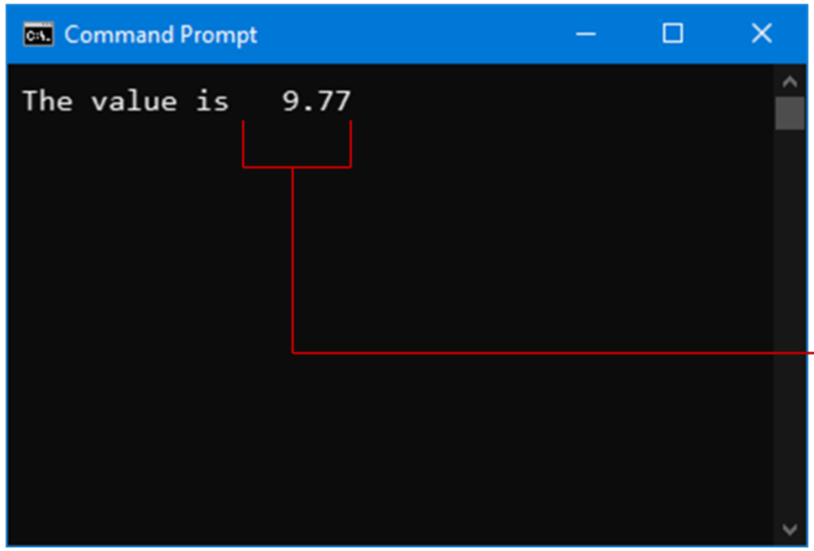
```
The value is      9
```

The `%6d` format specifier indicates the integer will appear in a field that is 6 spaces wide.

The `System.out.printf` Method

- Another example:

```
double number = 9.76891;
System.out.printf("The value is %6.2f\n", number);
```



```
The value is   9.77
```

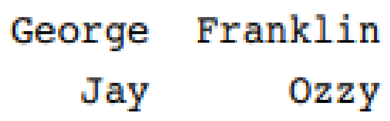
The `%6.2f` format specifier indicates the number will appear in a field that is 6 spaces wide and be rounded to 2 decimal places.

The `System.out.printf` Method

- You can also use a field width when printing strings. For example, look at the following code: The %10s format specifier prints a string in a field that is ten spaces wide.

```
String name1 = "George";
String name2 = "Franklin";
String name3 = "Jay";
String name4 = "Ozzy";
System.out.printf("%10s%10s\n", name1, name2);
System.out.printf("%10s%10s\n", name3, name4);
```

- Here is the output of the code:



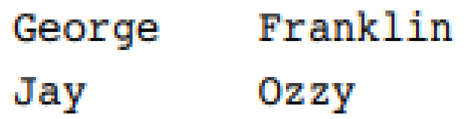
```
    George  Franklin
       Jay      Ozzy
```

The `System.out.printf` Method

- You can use the minus flag (-) to left-justify a string within its field. The following code demonstrates

```
String name1 = "George";
String name2 = "Franklin";
String name3 = "Jay";
String name4 = "Ozzy";
System.out.printf("%-10s%-10s\n", name1, name2);
System.out.printf("%-10s%-10s\n", name3, name4);
```

- Here is the output of the code:



```
George    Franklin
Jay       Ozzy
```

Programming Style

- Although Java has a strict syntax, whitespace characters are ignored by the compiler.
- The Java whitespace characters are:
  - space
  - tab
  - newline
  - carriage return
  - form feed

Indentation

- Programs should use proper indentation.
- Each block of code should be indented a few spaces from its surrounding block.
- Two to four spaces are sufficient.
- Tab characters should be avoided.
  - Tabs can vary in size between applications and devices.
  - Most programming text editors allow the user to replace the tab with spaces.

Try it out

- Write an application that calculates your final numeric grade for this class. Your program must ask the user to enter a student name and four floating point values which represent your scores for assignments , first midterm exam, second midterm exam, and final exam. Use printf to display your result

```
Scanner key = new Scanner(System.in);
String name;
double assignment, firstExam, secondExam, finalExam, finalGrade;
System.out.print("What is your name?  ");
name = key.nextLine();
System.out.print("what is your assignment score? ");
assignment = key.nextDouble();
System.out.print("what is your first exam score? ");
firstExam = key.nextDouble();
System.out.print("what is your second exam score? ");
secondExam = key.nextDouble();
System.out.print("what is your final exam score? ");
finalExam = key.nextDouble();
finalGrade = assignment + firstExam + secondExam + finalExam;
System.out.printf("Hello %s, your final score is %.2f\n", name, finalGrade);
```