COMP 202

Built-in Libraries and objects

CONTENTS:
•Introduction to objects
•Introduction to some basic Java libraries
- String
- Random
- other APIs

Classes and Objects

- An object is an entity that has data and methods
  - so far, we haven't seen data, only methods
  - examples:
    - teller machine objects that allow to withdraw money, make money transfers etc.
    - calculator objects that allows to perform addition, division, etc.
    - Scanner objects allow to read in data from the keyboard

Classes and Objects

- Each object belongs to a class
  - Examples
    - teller machine objects belong to the class *teller machine*
    - A scanner object belongs to the class `Scanner`
  - All objects of a class have the same methods
    - `Scanner` objects have `nextInt(), nextDouble(), …`
    - teller machine objects have withdraw,…
- A class can be seen as a blueprint which describes the general behavior of a set of similar objects
- That is, the class used to define an object can be thought of as the type of an object

About Libraries

- Programming languages come with many components:
  - Editor → To help you write the program
  - Compiler → To validate your syntax and convert you text file into an executable file
  - Syntax & Semantics → Rules that define how you write sentences and spell words in that language
  - Libraries → Pre-built classes and objects with methods that you can use in your program (this is nice since you do not need to write this code, someone else has done this for you). Libraries come with the compiler you buy or you can download them from the web (some free, some not).

System.out

- `System.out` is a variable that points to an object that belongs to the class `PrintStream`
- `System.out` has been predefined and set up for us as part of the Java standard class library.
- `println` is a method that has been implemented by someone else. We don't care what actions or statements are actually executed. We only need to know what it is supposed to do and what the input is.

```
System.out.println ("This is a long message to print in a single line.");
```

object variable — method — Information provided to the method (parameters)

The DOT Operator

- Java uses the period as a membership designation.
- For example:

```
System.out.println("My output");
```

The DOT Operator

- The DOT operator invokes the println method of the object to which the variable *out* points to.

Important Side Note!

Abstraction

- An *abstraction* hides (or ignores) details
- An object is abstract in that we don't really have to think about its internal details in order to use it
- We don't have to know the internals of the `println` method in order to invoke it;
  - we only need to know what it does but not the individual steps/actions needed to accomplish this
- Therefore, we can write complex software without having to know how parts of it actually work.

Strings vs. Characters

- Characters are a single letter or symbol:
  char x = 'a';
  char y = '%';
- Strings are many characters concatenated:
  String s = "Bob Smith";
- **char** is a built-in type
- **String** is a library object

The String Object

- Every character string is an object in Java.
  - "hello", "world", …
- The behavior of string objects is defined by the `String` class
- Every string literal, delimited by double quotation marks, represents a `String` object
- Since strings are so common, the Java programming language provides us with some form of syntactic sugar that allows us to use strings **nearly** in a way we use primitive data types.
- However, there are important differences between primitive data types and String objects and we have to aware of them.

String concatenation

- The *string concatenation operator* (+) is used to append one string to the end of another
  - `"hello" + " world"` results in a new string: `"hello world"`
- A string literal cannot be broken across two lines in a program

Facts.java

```
public class Facts
{
  public static void main(String[] args)
  {
      String firstName = "Bob";
      String lastName  = "Smith";
      String fullName;

      fullName = firstName + " " + lastName;

      System.out.println("His name is: " + fullName);
      System.out.println("Tel. prefix: " + 514);
      System.out.println("Tel. prefix: 514");
  }
}
```

What is the output?

The + operator

- The function that the + operator performs depends on the type of the information on which it operates
- If both operands are strings, it performs string concatenation
- if one is a string and one is a number
  - it converts the number into its string representation
  - then it concatenates both strings
- if one is a string and one is an arithmetic expression
  - it evaluates the arithmetic expression and converts the results into a string representation
  - then it concatenates both strings
- If both operands are numeric, it adds them
- The + operator is evaluated left to right
- Parentheses can be used to force the operation order

Addition.java

```
public class Addition
{
  public static void main(String[] args)
  {
      int x = 5, y = 2, sum = 0;
      String s = "The Sum is ";

      sum = x + y;
      s   = s + sum;

      System.out.println(s);
      System.out.println("The result is:" + x + y);
      Sustem.out.println("The result is:" + (x+y));
  }
}
```

What is the output?

Escape Sequences

- What if we wanted to print a double quote character?
- The following line would confuse the compiler because it would interpret the second quote as the end of the string

```
System.out.println("I said "Hello" to you.");
```

- An *escape sequence* is a series of characters that represents a special character
- An escape sequence begins with a backslash character (\), which indicates that the character that follows should be treated in a special way

```
System.out.println ("I said \"Hello\" to you.");
```


Escape Sequences

- Some Java escape sequences:

Escape Sequence	Meaning
`\b`	backspace
`\t`	tab
`\n`	newline
`\r`	carriage return
`\"`	double quote
`\'`	single quote
`\\`	backslash

Poem.java

```
pubic class Poem
{
  public static void main(String[] args)
  {
      System.out.println("Roses are red,");
      System.out.println("\t Violets are blue");
      System.out.println("\t When I look at you\n\t I do
  not know what to do");
  }
}
```

What does this print out?

Creating Objects

- A variable:
  - either holds a primitive type, or
  - it holds a *reference/pointer* to an object
- A class name can be used <u>like</u> a type to declare a *reference variable*

```
String name;
Scanner scan;
```

- No objects have been created with these declarations
- A reference variable holds the address of an object
- The object itself must be created separately

Creating Objects

- We use the `new` operator to create an object

```
name = new String ("Sparky the clown");
```

This calls the `String` *constructor*, which is a special method that sets up the object

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

- Creating an object is called *instantiation,* because an object is an *instance* of a particular class

String special

- Because strings are so common, we don't have to use the `new` operator to create a `String` object

```
name = "Sparky the clown";
```

- This is special syntax that only works for strings

Methods

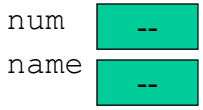
- Once an object has been instantiated, we can use the *dot operator* to invoke its methods

```
int length;
length = name.length();
length = scan.nextInt();
```

- Many methods have return values
  - `length()` returns an integer value
  - the return value can be assign to a variable
  - recall
    - `nextInt()` method of the Scanner class returns an integer value
    - `nextDouble()` method of the Scanner class returns a double value
- Many methods need input values
  - `System.out.println("Hello World");`
    - requires input of type String

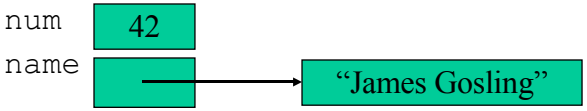
Primitive data vs. objects: Declaration

- `int num;`
  - creates a variable that holds an integer value
  - that is, allocates main memory cells which are accessed via the variable `num`
- `String name;`
  - creates an object variable
  - that is, allocates main memory cells which will contain the reference to an object (once the object is created)
- Initially, both variables don't contain any data



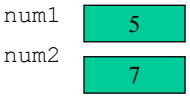
Primitive data vs. objects: Assignment I

- `num = 42;`
  - writes 42 into the cells referred to by `num`
- `name = new String("James Gosling");`
  - a string object is created; it uses some memory cells
  - the variable `name` holds the address of the memory cell that contains the string object
  - we say the object variable is a *pointer* to the real object

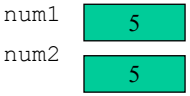


Primitive data: Assignment II

- `int num1 = 5;`
- `int num2 = 7;`

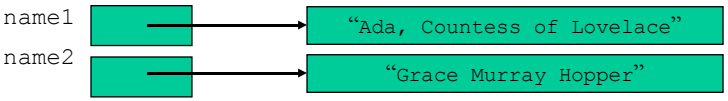


- `num2 = num1;`

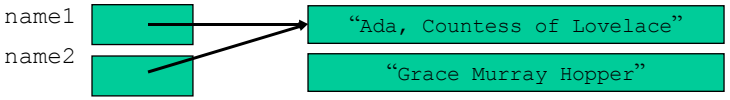


Objects: Assignment II

- `String name1 = "Ada, Countess of Lovelace";`
- `String name2 = "Grace Murray Hopper";`



- `name2 = name1;`



String Methods

- The `String` class has several methods that are useful for manipulating strings, ex:

Method	Meaning	Result
String(String str)	Creates a string object	Reference
char charAt(int index)	Get char at index	char
int compareTo(String str)	Is str the same?	0 if same
boolean equals(String str)	Is str the same?	boolean
boolean equalsIgnoreCase(String str)		
int length()	Number of characters	int
String replace(char oldChar, char newChar)	Find oldChar and replace	String
String substring(int offset, int endIndex)	Cutout part of string	String
String toLowerCase()	Convert to lowercase	String
String toUpperCase()	Convert to uppercase	String

- None of the methods changes the string object on which the method is called
- index: position of a character in a string (starting with 0)

StringMutate.java

```
public class StringMutate
{
  public static void main(String[] args)
  {
      String s = new String("This is a sentence");
      String s2 = s, s3 = "Bob";
      String temp;
      int answer;

      answer = s.compareTo(s2);
      temp   = s.substring(2,3);
      s3     = s3.replace('b','c');
  }
}
```

What is in answer, temp and s3?

Class Libraries

- A *class library* is a collection of classes that we can use when developing programs
- There is a *Java standard class library* that is part of any Java development environment
- These classes are not part of the Java language per se, but we rely on them heavily
- The `System` class and the `String` class are part of the Java standard class library
- Other class libraries can be obtained through third party vendors, or you can create them yourself

Packages

- The classes of the Java standard class library are organized into packages
- Some of the packages in the standard class library are:

Package	Purpose
`java.lang`	**General support (e.g., contains System and String)**
`java.applet`	**Creating applets for the web**
`(java.awt`	***Graphics and graphical user interfaces)***
`javax.swing`	**Additional graphics capabilities and components**
`java.net`	**Network communication**
`java.util`	**Utilities**

The import Declaration

- When you want to use a class from a package, you could use its *fully qualified name*

```
java.util.Scanner
```

- Or you can *import* the class, then just use the class name

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

- To import all classes in a particular package, you can use the * wildcard character

```
import java.util.*;
```


No import for `java.lang`

- All classes of the `java.lang` package are automatically imported into all programs
- That's why we didn't have to explicitly import the `System` or `String` classes in earlier programs

The Scanner Class

- The `Scanner` class is part of the Java standard class library
- Part of the `java.util` package
- Contains several methods for reading input values for various types from various sources (keyboard, file)
- A `Scanner` object assumes that white space (delimiters) separates elements of the input (tokens)

The Scanner class so far

- `Scanner scan = new Scanner(System.in);` → Keyboard
  - `scan` is an object variable
  - Constructor takes as input the source we want to read;
    - keyboard so far
  - after creation of Scanner object, `scan` points to this object
- `int number1 = scan.nextInt();`
- `double number2 = scan.nextDouble();`

Some methods of the Scanner class

- `Scanner (InputStream source)`
- `Scanner (File source)`
- `Scanner (String source)`
- `String next ()`
- `String nextLine ()`
- `double nextDouble ()`
- `float nextFloat ()`
- `int nextInt ()`
- `long nextLong ()`
- `short nextShort ()`
- `String nextBoolean ()`
- `Boolean hasNext ()`

Printing names

```
import java.util.Scanner;
 public class PrintNames
 {
   public static void main(String[] args)
   {
     String firstName, lastName;
     boolean done = false;
     Scanner scan = new Scanner(System.in);

     while(!done)
     {
         System.out.println("enter first name (\"done\" exits)");
         firstName = scan.nextLine();
         if (firstName.equals("done"))
           done = true;
         else
         {
           System.out.println("enter last name:");
           lastName = scan.nextLine();
           System.out.println("The full name is " + firstName
                                    + " " + lastName);
         }
     }
   }
 }
```

What does this print out?

Random Numbers

- *pseudo-random* number generator
  - performs a series of complicated calculations, based on an initial seed value, and produces a number
  - these numbers *appear* to be randomly selected
- `Random()`
  - constructor
- `float nextFloat()`
  - returns a random number between 0.0 (incl.) and 1.0 (excl.)
- `int nextInt()`
  - returns a random number that ranges over all possible int values (positive and negative)
- `int nextInt(int num)`
  - returns a random number in the range 0 to num-1
- uniform distribution, though `double nextGaussian()` is available too

RandomNumber.java

```
import java.util.Random;

public class RandomNumber
{
  public static void main(String args[])
  {
      Random generator = new Random();
      int num1;
      float num2;

      num1 = generator.nextInt(10); // 0 - 9
      num2 = generator.nextFloat(); // 0.0 - 1.0
  }
}
```

Class Methods

- Some methods can be invoked through the class name, instead of through an object of the class
  - that is, we do not need to declare an object variable and create an object before calling these methods
- These methods are called *class methods* or *static methods*
- The `Math` class contains many static methods, providing various mathematical functions, such as absolute value, trigonometry functions, square root, power, PI (see Chapter 3 for more methods)

```
temp = Math.cos(90) + Math.sqrt(delta);
```


Circle

```
import java.util.Scanner;
public class Circle
{
  public static void main (String [] args)
  {
    double radius, circumference, area;
    Scanner scan = new Scanner(System.in);

    // read in the radius
    System.out.println("Enter radius:");
    radius = scan.nextDouble();

    // perform calculation
    circumference = 2 * Math.PI * radius;
    area = Math.pow(radius,2) * Math.PI;
    System.out.println("The circumference is: " + circumference);
    System.out.println("The area is: " + area);
  }
}
```


Formatting Output

- The `DecimalFormat` class can be used to format a floating point value in generic ways
- For example, you can specify that the number be printed to three decimal places
- First you have to create an object of the class `DecimalFormat`
  - for that you use the constructor of the `DecimalFormat` class
  - The constructor takes a string that represents a pattern (e.g., indicating three decimal places)
- The `DecimalFormat` object has a method called `format` that takes as input a floating point number and returns a string with the specified information in the appropriate format

Circle formatted

```
import java.util.Scanner;
import java.text.DecimalFormat;
public class Circle
{
  public static void main (String [] args)
  {
    double radius, circumference, area;
    Scanner scan = new Scanner(System.in);
    DecimalFormat f = new DecimalFormat("0.###");

    // read in the radius
    System.out.println("Enter radius:");
    radius = scan.nextDouble();

    // perform calculation
    circumference = 2 * Math.PI * radius;
    area = Math.pow(radius,2) * Math.PI;
    System.out.println("The circumference is: " + f.format(circumference));
    System.out.println("The area is: " + f.format(area));
  }
}
```

Wrapper classes

- Two categories of data in Java : primitive data and objects.
- We use *wrapper classes* to manage primitive data as objects
- Each wrapper class represents a particular primitive type

```
Integer everest = new Integer(8850);
```

- We have just « *wrapped* » the primitive integer value `8850` into an object referenced by the `everest` variable.
- All wrapper classes are part of the `java.lang` package: `Byte`, `Short`, `Integer`, `Long`, `Float`, `Double`, `Character`, `Boolean`, `Void`.

Some methods of the Integer class

- Integer (int value)

- byte byteValue()
- double doubleValue()
- float floatValue()
- int intValue()
- long longValue()

Example:

```
String s = "500";
int number;

number = Integer.parseInt(s);
```

- static int parseInt (String str)
- static String toBinaryString(int num)

Formatting Output

- The `NumberFormat` class has static methods that return a formatter object

```
getCurrencyInstance()
getPercentInstance()
```

- Each formatter object has a method called `format` that returns a string with the specified information in the appropriate format

Purchase.java

```
import java.util.Scanner;
import java.text.NumberFormat;
class Purchase
{
  public static void main(String args[])
  {
      double amount, total;
      Scanner in = new Scanner(System.in);
      NumberFormat f = NumberFormat.getCurrencyInstance();

      amount = in.nextDouble();
      total = ((amount + (amount * 0.06)) * 0.075) + amount;

      System.out.println("Please pay: " + f.format(total));
  }
}
```

What does this print out?