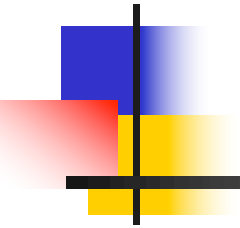
ADVANCED PROGRAMMING



JAVA BASIC 4

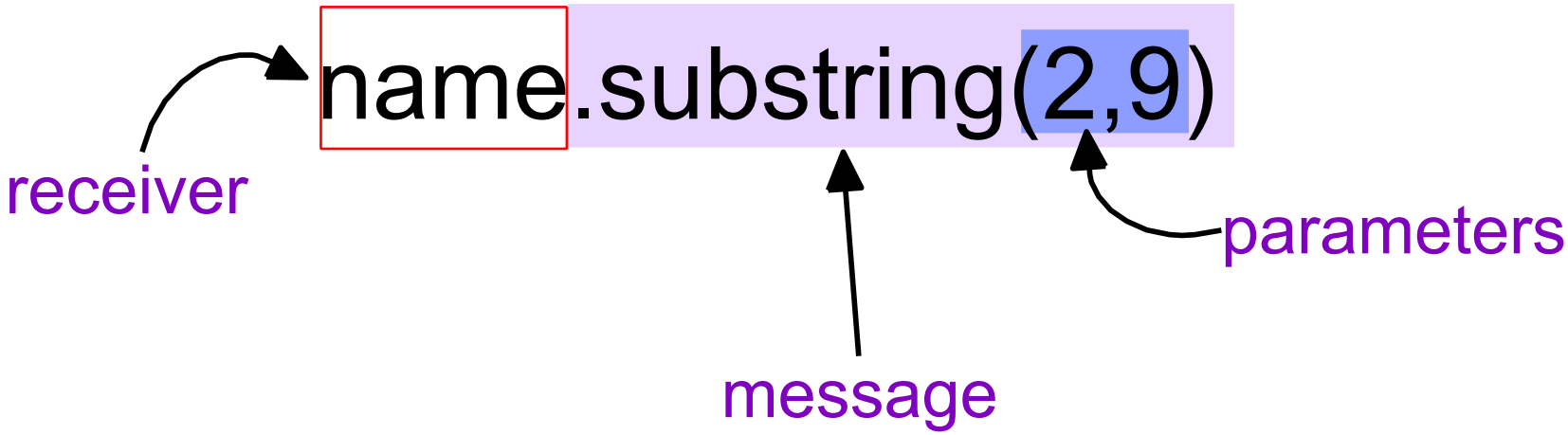
BUILT IN OBJECTS

Section Goals

- Continue learning the basic syntax of Java
- Understand how objects are used
- Provide an introduction to built-in Java classes:
  - String & StringBuffer
  - Wrapper classes
- Array
- ArrayList and Map
- Format output using Escape Sequence

Objects and Messages

- Objects provide more complex behavior than primitives
- Objects respond to messages
  - Use the dot "." operator



Declaring and Initializing Objects

- Just like primitives and arrays, objects must be declared before they can be used
  - The declaration requires the type of the object
  - Use '=' for assignment (including initialization)
  - Initialization of an object often uses the `new` operator
  - An object can be initialized to `null`
- Arrays of objects are declared just like arrays of primitives
  - Arrays of objects default to initialization with `null`
- Examples:

```
Employee emp1 = new Employee(123456);
Employee emp2;
emp2 = emp1;
Department dept[] = new Department[100];
Test[] t = {new Test(1), new Test(2)};
```

Identity

- The == operator
  - Tests for exact object identity
  - Checks whether two variables reference the same object
  - For primitive types, checks for equal values

```
Employee a = new Employee(1);
Employee b = new Employee(1);
if (a == b)... // false
```

```
Employee a = new Employee(1);
Employee b = a;
if (a == b)... // true
```

```
int a = 1;
int b = 1;
if (a == b)... // true
```

Wrapper Classes

- Primitives have no associated methods
- Wrapper classes:
  - Encapsulate primitives
  - Provide methods to work on them
  - Are included as part of the base Java API

Primitive Type	Wrapper Class
boolean	Boolean
byte	Byte
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short

Using Wrapper Classes

```
double number = Double.parseDouble("42.76");
```

```
String hex = Integer.toHexString(42);
```

```
double value = new Integer("1234").doubleValue();
```

```
String input = "test 1-2-3";
int output = 0;
for (int index = 0; index < input.length(); index++) {
   char c = input.charAt(index);
   if (Character.isDigit(c))
      output = output * 10 + Character.digit(c, 10);
}
System.out.println(output); // 123
```

Characters and Strings

- **String** — A class for working with immutable (unchanging) data composed of multiple characters
- **StringBuffer** — A class for storing and manipulating mutable data composed of multiple characters. This class is safe for use in a multi-threaded environment
- **StringBuilder** — A faster, drop-in replacement for **StringBuffer**, designed for use by a single thread only

Character's methods

Method	Description
boolean isLetter(char ch) boolean isDigit(char ch)	Determines whether the specified char value is a letter or a digit, respectively.
boolean isWhiteSpace(char ch)	Determines whether the specified char value is white space according to the Java platform.
boolean isUpperCase(char ch) boolean isLowerCase(char ch)	Determines whether the specified char value is upper- or lowercase, respectively.
char toUpperCase(char ch) char toLowerCase(char ch)	Returns the upper- or lowercase form of the specified char value.
toString(char ch)	Returns a String object representing the specified character value.

Strings

- Any number of characters between double quotes is a String:

```
String a = "A String";
String b = "";
```

- String can be initialized in other ways:

```
String c = new String();
String d = new String("Another String");
String e = String.valueOf(1.23);
String f = null;
```

Concatenating Strings

- The + operator concatenates Strings:

```
String a = "This" + " is a " + "String";
```

- Primitive types used in a call to println are automatically converted to Strings

```
System.out.println("answer = " + 1 + 2 + 3);
System.out.println("answer = " + (1 + 2 + 3));
```

- Do you get the same output from the above examples?

Comparing Strings

- `oneString.equals(anotherString)`
  - Tests for equivalence
  - Returns `true` or `false`
- `oneString.equalsIgnoreCase(anotherString)`
  - Case insensitive test for equivalence
  - Returns `true` or `false`
- `oneString == anotherString` is problematic

```
String name = "Joe";
if ("Joe".equals(name))
   name += " Smith";
```

```
boolean same = "Joe".equalsIgnoreCase("joe");
```

String Messages

- Strings are objects; objects respond to messages
  - Use the dot (.) operator to send a message
  - String is a class, with methods (more later)

```
String name = "Joe Smith";
name.toLowerCase();     // "joe smith"
name.toUpperCase();     // "JOE SMITH"
" Joe Smith ".trim();   // "Joe Smith"
"Joe Smith".indexOf('e'); // 2
"Joe Smith".length();     // 9
"Joe Smith".charAt(5);    // 'm'
"Joe Smith".substring(5);   // "mith"
"Joe Smith".substring(2,5); // "e S"
```

StringBuffer

- StringBuffer is a more efficient mechanism for building strings
  - String concatenation
    - Can get very expensive
    - Is converted by most compilers into a StringBuffer implementation
  - If building a simple String, just concatenate
  - If building a String through a loop, use a StringBuffer

```
StringBuffer buffer = new StringBuffer(15);
buffer.append("This is ");
buffer.append("String");
buffer.insert(7, " a");
buffer.append('.');
System.out.println(buffer.length()); // 17
System.out.println(buffer.capacity()); // 32
String output = buffer.toString() ;
System.out.println(output); // "This is a String."
```

Creating Strings

- **String(byte[] bytes)**
  Constructs a new String by decoding the specified array of bytes using the platform's default charset.
- **String(byte[] bytes, String charsetName)**
  Constructs a new String by decoding the specified array of bytes using the specified charset.
- **String(char[] value)**
  Allocates a new String so that it represents the sequence of characters currently contained in the character array argument.
- **String(char[] value, int offset, int count)**
  Allocates a new String that contains characters from a subarray of the character array argument.

Creating Strings

- **String(int[] codePoints, int offset, int count)**
  Allocates a new String that contains characters from a subarray of the Unicode code point array argument.
- **String(String original)**
  Initializes a newly created String object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.
- **String(StringBuffer buffer)**
  Allocates a new string that contains the sequence of characters currently contained in the string buffer argument.
- **String(StringBuilder builder)**
  Allocates a new string that contains the sequence of characters currently contained in the string builder argument.

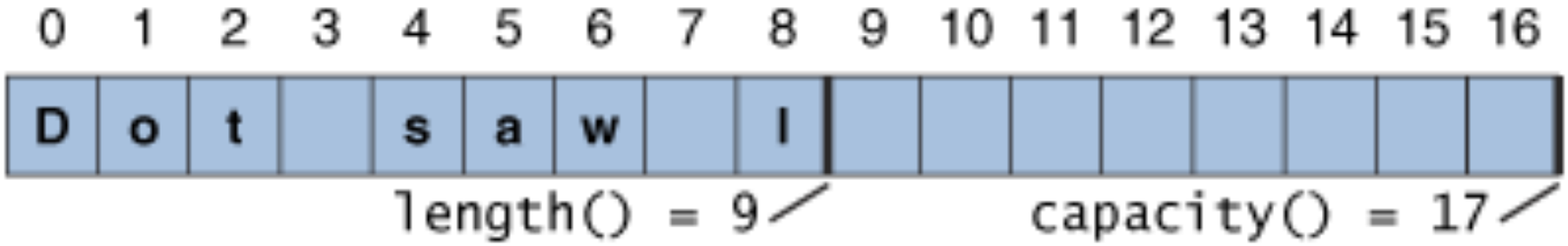
Creating StringBuilder

- **StringBuilder()**
  Constructs a string builder with no characters in it and an initial capacity of 16 characters.
- StringBuilder(CharSequence seq)
  Constructs a string builder that contains the same characters as the specified CharSequence.
- **StringBuilder(int capacity)**
  Constructs a string builder with no characters in it and an initial capacity specified by the capacity argument.
- **StringBuilder(String str)**
  Constructs a string builder initialized to the contents of the specified string.

Getting the Length

```
public class StringsDemo {
    public static void main(String[] args) {
        String palindrome = "Dot saw I was Tod";
        int len = palindrome.length();
        StringBuilder dest = new StringBuilder(len);
        for (int i = (len - 1); i >= 0; i--) {
            dest.append(palindrome.charAt(i));
        }
        System.out.format("%s%n", dest.toString());
    }
}
```

- The **`StringBuilder`** and **`StringBuffer`** classes have a method called **`capacity`**, which returns the amount of space allocated rather than the amount of space used.

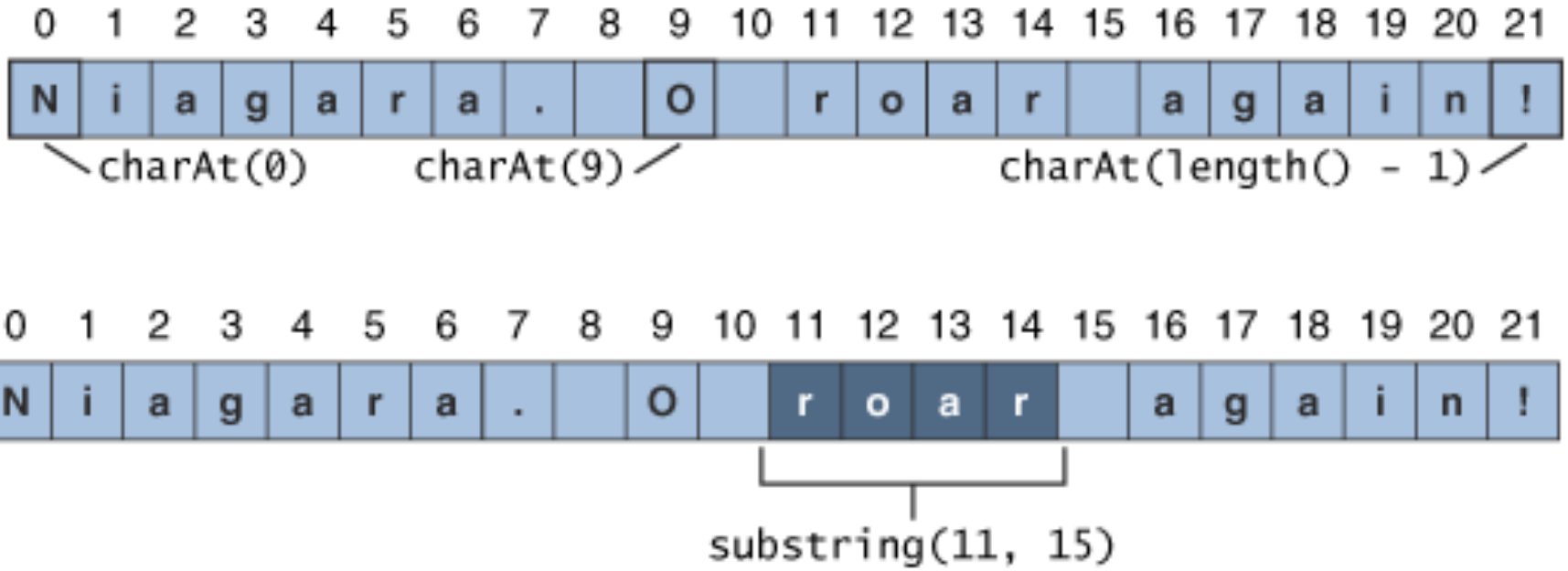


Getting Characters by Index

- **public char charAt(int index)**
  Returns the char value in this sequence at the specified index. The first char value is at index 0.
- **public String substring(int start)**
  Returns a substring begins at the specified index and extends to the end of this sequence.
- **public String substring(int start, int end)**
  Returns a substring begins at the specified start and extends to the character at index end - 1.

Getting Characters by Index

- String anotherPalindrome = "Niagara. O roar again!";
- char aChar = anotherPalindrome.charAt(9);
- String roar = anotherPalindrome.substring(11, 15);



Searching for a Character

Method	Description
int indexOf(char) int lastIndexOf(char)	Returns the index of the first (last) occurrence of the specified character.
int indexOf(char, int) int lastIndexOf(char, int)	Returns the index of the first (last) occurrence of the specified character, searching forward (backward) from the specified index.

- The **StringBuffer** and **StringBuilder** classes do not support the **indexOf** or the **lastIndexOf** methods. If you need to use these methods on either one of these objects, first convert to a string by using the **toString** method

Searching for a Substring

Method	Description
int indexOf(String) int lastIndexOf(String)	Returns the index of the first (last) occurrence of the specified string.
int indexOf(String, int) int lastIndexOf(String, int)	Returns the index of the first (last) occurrence of the specified string, searching forward (backward) from the specified index.
boolean contains(CharSequence)	Returns true if the string contains the specified character sequence.

FilenameDemo

```
public class FilenameDemo {
  public static void main(String[] args) {
    final String FPATH = "/home/mem/index.html";
    Filename myHomePage = new
                                    Filename(FPATH,'/', '.');
    System.out.println("Extension = " +

  myHomePage.extension());
    System.out.println("Filename = " +
                                     myHomePage.filename());
    System.out.println("Path = " +
                                       myHomePage.path());
    }
}
```

Extension = html
Filename = index
Path = /home/mem

FileName

```
public class Filename {
    private String fullPath;
    private char pathSeparator,  extensionSeparator;

    public Filename(String str, char sep, char ext) {
        fullPath = str;
        pathSeparator = sep;
        extensionSeparator = ext;
    }

    public String extension() {
        int dot = fullPath.lastIndexOf(extensionSeparator);
        return fullPath.substring(dot + 1);
    }
```

FileName

```
public String filename() {
        int dot = fullPath.lastIndexOf(extensionSeparator);
        int sep = fullPath.lastIndexOf(pathSeparator);
        return fullPath.substring(sep + 1, dot);
    }

    public String path() {
        int sep = fullPath.lastIndexOf(pathSeparator);
        return fullPath.substring(0, sep);
    }
}
```

Comparing Strings and Portions of Strings

Methods in the String Class for Comparing Strings	
Method	Description
boolean endsWith(String) boolean startsWith(String) boolean startsWith(String, int)	Returns true if this string ends with or begins with the substring specified as an argument to the method. The integer argument, when present, indicates the offset within the original string at which to begin looking.
int compareTo(String) int compareToIgnoreCase (String)	Compares two strings and returns an integer indicating whether this string is greater than (result is > 0), equal to (result is = 0), or less than (result is < 0) the argument. The compareToIgnoreCase method ignores case; thus, "a" and "A" are considered equal.

Comparing Strings and Portions of Strings

Methods in the String Class for Comparing Strings

Method	Description
boolean equals(Object) boolean equalsIgnoreCase(String) boolean contentEquals(CharSequence)	Returns true if this string contains the same sequence of characters as the argument. The Object argument is converted to a string before the comparison takes place. The equalsIgnoreCase method ignores case; thus, "a" and "A" are considered equal.

Modifying StringBuffers and StringBuilders

Methods for Modifying a String Buffer	
Method	Description
StringBuffer append(boolean) StringBuffer append(char) StringBuffer append(char[]) StringBuffer append(char[], int, int) StringBuffer append(double) StringBuffer append(float) StringBuffer append(int) StringBuffer append(long) StringBuffer append(Object) StringBuffer append(String)	Appends the argument to this string buffer. The data is converted to a string before the append operation takes place.
StringBuffer delete(int, int) StringBuffer deleteCharAt(int)	Deletes the specified character(s) in this string buffer.

Modifying String Buffers and String Builders

Methods for Modifying a String Buffer	
Method	Description
StringBuffer insert(int, boolean) StringBuffer insert(int, char) StringBuffer insert(int, char[]) StringBuffer insert(int, char[], int, int) StringBuffer insert(int, double) StringBuffer insert(int, float) StringBuffer insert(int, int) StringBuffer insert(int, long) StringBuffer insert(int, Object) StringBuffer insert(int, String)	Inserts the second argument into the string buffer. The first integer argument indicates the index before which the data is to be inserted. The data is converted to a string before the insert operation takes place.
StringBuffer replace(int, int, String) void setCharAt(int, char)	Replaces the specified character(s) in this string buffer.
StringBuffer reverse()	Reverses the sequence of characters in this string buffer.

Lists

- Problem
  - You want to make an ordered list of objects. But, even after you get the first few elements, you don't know how many more you will have.
    - Thus, you can't use an array, since the size of arrays must be known at the time that you allocate it.
- Solution
  - Use `ArrayList` or `LinkedList`: they stretch as you add elements to them
- Notes
  - The two options give the same results for the same operations, but differ in performance

ArrayList & LinkedList

- Summary of operations
  - Create empty list
    `new ArrayList<Type>()` or
    `new LinkedList<Type>()`
  - Note that you need "`import java.util.*;`" at the top of file
  - Add entry to end
    `add(value)` (adds to end) or `add(index, value)`
  - Retrieve n^{th} element
    `get(index)`
  - Check if element exists in list
    `contains(element)`
  - Remove element
    `remove(index)` or `remove(element)`
  - Find the number of elements
    `size()`

ArrayList Example

```
import java.util.*; // Don't forget this import
public class ListTest2 {
   public static void main(String[] args) {
      List<String> entries = new ArrayList<String>();
      double d;
      while ((d = Math.random()) > 0.1) {
         entries.add("Value: " + d);
      }

      for (String entry : entries) {
         System.out.println(entry);
      }
   }
}
```

This tells Java your list will contain only strings.

```
Value: 0.6374760850618444
Value: 0.9159907384916878
Value: 0.8093728146584014
Value: 0.7177611068808302
Value: 0.9751541794430284
Value: 0.2655587762679209
Value: 0.3135791999033012
Value: 0.44624152771013836
Value: 0.7585420756498766
```

HashMap

- HashMap provides simple lookup table
  - Use "put" to store data

```
Map<String,Person> employees =
                        new HashMap<String,Person>();
```

- The table keys will be Strings; the associated values will be Persons.

```
Person p1 = new Person("a1234", "Larry",
"Ellison");
employees.put(p1.getEmployeeId(), p1);
```

  - Use "get" to retrieve data

```
Person p = employees.get("a1234");
```

    - Returns null if no match

Assignment 1

- Write a static method which takes an ArrayList of integers and an integer and changes the ArrayList destructively to remove all occurrences of the integer argument. So if the integer argument is 9 and the ArrayList is

21	9	13	9	5	10	19	36	9	21

the `ArrayList` should be changed to:

21	13	5	10	19	36	21

Formatting Output

Formatted Output: printf

- Takes a variable number of arguments
  - System.out.printf("Formatting String", arg1, arg2, …);
- Advantages
  - Lets you insert values into output without much clumsier String concatenation.
  - Lets you control the width of results so things line up
  - Lets you control the number of digits after the decimal point in numbers, for consistent-looking output
- Very similar to C/C++ printf function
  - If you know printf in C/C++, you can probably use Java's printf immediately without reading any documentation
    - Although some additions in time formatting and locales
  - Use String.format to get the equivalent of C's sprintf

printf vs. println

- General idea
  - Each %s entry in formatting string is replaced by next argument in argument list. %n means newline.
- Example

```
public static void printSomeStrings() {
   String firstName = "John";
   String lastName = "Doe";
   int numPets = 7;
   String petType = "chickens";
   System.out.printf("%s %s has %s %s.%n",
             firstName, lastName, numPets, petType);
   System.out.println(firstName + " " + lastName +
             " has " + numPets + " " + petType + ".");
}
```

Output

```
John Doe has 7 chickens.
John Doe has 7 chickens.
```

Controlling Formatting

- Different flags
  - %s for strings, %f for floats/doubles, %t for dates, etc.
    - Unlike in C/C++, you can use %s for any type (even nums)
- Various extra entries can be inserted
  - To control width, number of digits, commas, justification, type of date format, and more
- Complete details
  - printf uses mini-language
- Most common errors
  - Using + instead of , between arguments (printf uses varargs)
  - Forgetting to add %n at the end if you want a newline (not automatic)

Printf Formatting Options

	Stands For	Options	Example
%s	String. Can output any data type. If arg is Object, toString is called.	%width**s** Gives min num of chars. Spaces added to left if needed.	`printf("%8s", "Hi")` Outputs " Hi"
%d	Decimal. Outputs whole number in base 10. Also %x and %o for hex and octal.	%width**d** %,width**d** Gives min width; inserts commas.	`printf("%,9d", 1234)` Outputs " 1,234"
%f	Floating point. Lets you line up decimal point and control precision.	%width.precision**f** %,width.precision**f** width includes comma and decimal point.	`printf("%6.2f", Math.PI)` Outputs " 3.14"
%t x	Time (or date). %tA for day, %tB for month, %tY for year, and many more.	Date now = new Date(); printf("%tA, %tB ,%tY", now, now, now) Outputs "Thursday, November 17, 2005"	
%n	Outputs OS-specific end of line (linefeed on Linux, CR/LF pair on Windows)		

printf Example

```
public static void printSomeSalaries() {
   CEO[] softwareCEOs = {
                  new CEO("Steve Jobs", 3.1234),
                  new CEO("Scott McNealy", 45.5678),
                  new CEO("Jeff Bezos", 567.982323),
                  new CEO("Larry Ellison", 6789.0),
                  new CEO("Bill Gates", 78901234567890.12)};
   System.out.println("SALARIES:");
   for (CEO ceo : softwareCEOs) {
      System.out.printf("%15s: $%,8.2f%n",
                  ceo.getName(), ceo.getSalary());
   }
}
```

Output

```
SALARIES:
Steve Jobs: $ 3.12
Scott McNealy: $ 45.57
Jeff Bezos: $ 567.98
Larry Ellison: $6,789.00
25 Bill Gates: $78,901,234,567,890.12
```

printf Example

```
public class CEO {
    private String name;
    private double salary; // In billions

    public CEO(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }

    public String getName() { return name; }

    public double getSalary() { return salary; }
}
```

Assignment 2 – làm tại lớp

- Hãy hiện thực phương thức sắp xếp tăng dần cho một mảng một chiều bất kỳ?
- Ví dụ: các loại mảg 1 chiều có thể đưa vào test dưới các dạng sau:
  - Integer[] a={3,1,5,2,7};
  - Double[] a={3,1,5,2,7};
  - Float[] a={3.6f,1.2f,5.6f,2.8f,7.7f}

Assignment 3

- Viết chương trình thực hiện các yêu cầu sau:

1. Nhập 1 chuỗi bất kỳ từ bàn phím, sau đó hiển thị độ dài của chuỗi vừa nhập vào và các ký tự có trong chuỗi đó (một ký tự chỉ hiển thị 1 lần)
2. Viết chương trình thực hiện các yêu cầu sau:
    1. Dùng HashSet có kiểu dữ liệu là String. Sau đó thêm vào HashSet này tên các loại cái cây được nhập từ bàn phím.
    2. Nhập vào 1 loại tên trái cây và kiểm tra loại này có tồn tại trong HashSet không. Nếu có thông báo tìm thấy, nếu không thông báo không tìm thấy
    3. Xoá một loại tên trái cây bất kỳ trong HashSet cà hiện thị các phần tử còn lại