Class 3

String Builder class (This should be new to everyone)

Because strings are immutable, programs with heavy concatenation can be pretty inefficient because a new string must be created for each instance of concatenation.

Java also provides the StringBuilder class. Unlike String objects, StringBuilder objects can be changes. In other words, a StringBuilder object is like a String but it is NOT immutable.

Many of the methods are the same as those of the String class. For example, the StringBuilder class has methods such as length() or charAt(i)

Unlike String objects StringBuilder objects can be altered. Strings are immutatable; StringBuilders are not. A StringBuilder is like a String that you can change.

Here is how you instantiate or create StringBuilder objects.

When you instantiate or create a StringBuilder object a *buffer* (section of memory) is created for the characters of the StringBuilder object. A program can change the characters in the buffer. In other words, a StringBuilder object, unlike a String, is NOT immutable.

Example:

StringBuilder sb = new StringBuilder()	Default buffer holds 16 characters
StringBuilder sb = new StringBuilder(100)	Initially buffer holds 100 characters
StringBuilder sb = new StringBuilder("Hello")	Initially buffer is size 6 and contains the characters of "Hello" (sb→ "Hello")

The buffer can expand as needed and the characters in the buffer can be changed.

StringBuilderMethods

Method	**Explanation (sb refers to a StringBuilder)**	**Example**
StringBuilder append(String s) StringBuilder append(char c) StringBuilder append(StringBuilder s)	sb.append(x) appends *x* to the end of sb and returns a reference to the **altered** StringBuilder object.	StringBuilder s = new StringBuilder("New"); s.append(" York"); returns StringBuilder("New York") **and** alters s
char charAt(int i)	sb.charAt(i) returns the character at position i. StringBuilder character sequences are indexed from 0	StringBuilder s = new StringBuilder("Iowa"); char ch = sb.charAt(3); ch has the value 'a'

StringBuilder delete(int start, int end)	sb.delete(start, end) removes the characters from position start to position end -1 and returns a reference to the altered StringBuilder object.	StringBuilder s= new StringBuilder("Delaware"); s.delete(2,6); returns StringBuilder("Dere") and alters s
StringBuilder deleteCharAt(int i)	sb.deleteCharAt(i) removes the character at index i and returns a reference to the altered StringBuilder object.	StringBuilder s = new StringBuilder("Maine"); s.deleteCharAt(1); returns StringBuilder("Mine") and alters s
int indexOf(String s)	sb.indexOf(s) returns the index of the first occurrence of s in sb. If s is not a substring of sb, returns –1.	StringBuilder s = new StringBuilder("Florida"); int x = s.indexOf("or"); x has the value 2
int indexOf(String s, int from)	sb.indexOf(s, from) returns the index of the first occurrence of s in sb starting at index from. If s is not a substring of sb, returns –1.	StringBuilder s = new StringBuilder("Mississippi"); int x = s.indexOf("is",2); x has the value 4
StringBuilder insert(int index, String s) StringBuilder insert(int index, char ch)	sb.insert(index, s) inserts s into sb at position index.	StringBuilder s = new StringBuilder("Oo"); s.insert(1, "hi); returns StringBuilder("Ohio") and alters s
int length()	sb.length() returns the number of characters in sb.	StringBuilder s = new StringBuilder("Vermont"); s.length returns 7
StringBuilder replace(int start, int end, String s)	sb.replace(start, end,s) replaces all characters from start to end-1 with s and returns a reference to the altered StringBuilder object.	StringBuilder s = new StringBuilder("Texas"); s.replace(1,4,"axe") returns "Taxes" and alters s
StringBuilder reverse()	sb.reverse() reverses the order of the characters of sb and returns a reference to the altered StringBuilder object.	StringBuilder s = new StringBuilder("Utah"); s.reverse() returns StringBuilder("hatU") and alters s
String substring(int index)	s.substring(index) returns the substring of s consisting of all characters with index greater than or equal to index. Notice that the method returns a String reference.	StringBuilder sb = new StringBuilder("New Jersey"); sb.substring(4) returns "Jersey"
String substring(int start, int end) Notice that the method returns a String reference.	s.substring(start, end) returns the substring of s consisting of all characters with index greater than or equal to start and strictly less than end.	StringBuilder sb = new StringBuilder("New Jersey"); sb.substring(0,3) returns "New"

String toString()	sb.toString() returns a String representation of the characters of sb.	StringBuilder s = new StringBuilder("Illinois"); String str =s.toString(); str refers to the **String** object "Illinois"

The Equals(..) Method of StringBuilder

Note: **Unlike the String class the equals of the StringBuilder class compares REFERENCES not characters. A reference is a memory address**

StringBuilder a = new StringBuilder("Sheldon"); StringBuilder b = new StringBuider("Sheldon");	**a ⟶ "Sheldon"** **b ⟶ "Sheldon"**

a.equals(b) is false because the equals of StringBuilder compares references
How can we compare characters of a StringBuilder object?

String s = a.toString(); // returns a String version of a
String t = toString();

s.equals(t) returns true --> String equals compares characters
Here we are using the **equals() of the String** class, which compare characters
Or you can alao use
(a.toString()).equals(b.toString())

Example:
Two of the methods of the StringBuilder class are
append(String s) and append(char c) // adds a String or character to the end

StringBuilder sb = new StringBuilder("Hello"); // sb →"Hello"
Sb.append(" Newman"); // sb → "Hello Newman"

This code does NOT create a new StringBuilder object but adds " Newman" to the end of the StringBuilder "Hello"

Example:

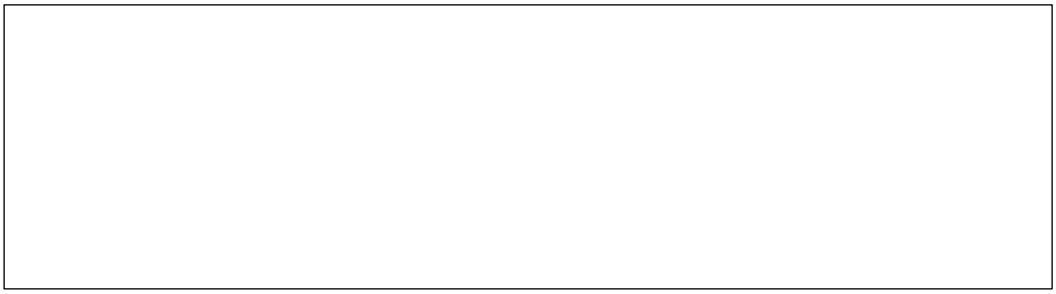
StringBuilder sb = new StringBuilder("Kramer");	**sb ⟶ "Kramer"**
sb.delete(1,4); // remove characters (1) through (3)	**sb ⟶ "Ker"**
sb.delete(sb.append("mit Frog");	**sb ⟶ "Kermit Frog"**
sb.insert(8,"The ");//insert at position 8	**sb ⟶ "Kermit The Frog"**
Sb.reverse();	**sb ⟶ gorF ehT timreK**

In each case, the StringBuilder object is changed. A new one is not created.

Here is an exercise that compares Strin and StringBuilder methods

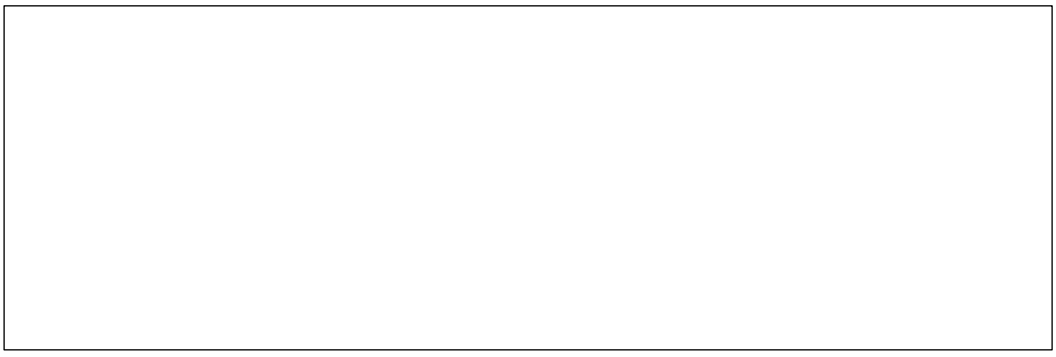
```
public class Strings
{
  public static void main(String[] args)
  {
    String alphabet         = new String("abcdefghijklmnopqustuvwxyz");
    StringBuilder alphabet1 = new StringBuilder("abcdefghijklmnopqustuvwxyz");
```

// removes "efghi" from the alphabet using String – fill in the code



// removes "efghi" from the alphabet1 using StringBuilder

//reverse a sequence of characters on alphabet using String



//reverse a sequence of characters in alphabet1 using StringBuilder

alphabet1 = new StringBuilder("abcdefghijklmnopqustuvwxyz"); **// alphabet1 had been changed**

// What is the output?

```
String a = new String("Homer");
String b = new String ("Homer");
```

// == vs equals for String and StringBuilder

```
System.out.println("Using equals with String : "+ a.equals(b));
System.out.println("Using == with String : "+ a == b);
```

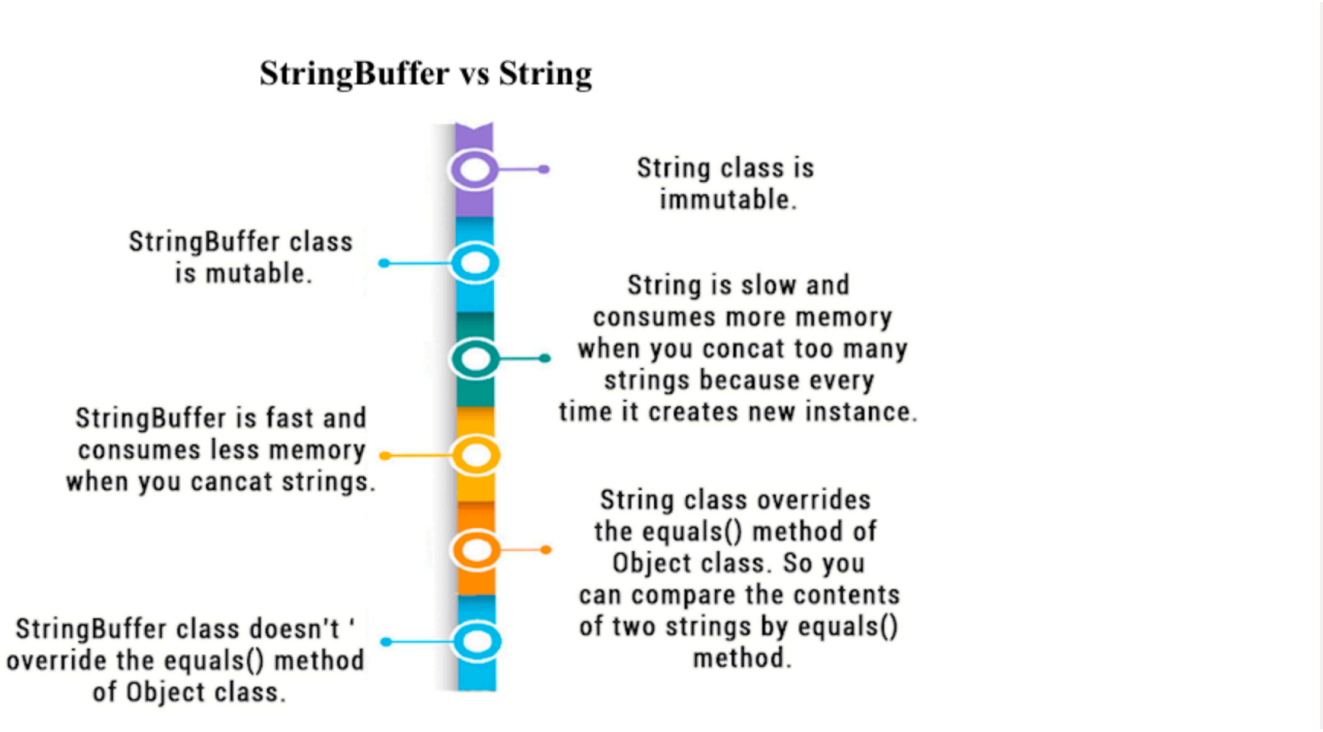
```
StringBuilder c = new StringBuilder("Homer");
StringBuilder d = new StringBuilder ("Homer");

System.out.println("Using equals with StringBuilder : "+ c.equals(d));
System.out.println("Using == with StringBuilder : "+ (c == d));
System.out.println("Using equals with StringBuilder and toString() : "+
                                                    c.toString().equals(d.toString()));
```

```
 }
}
```

Here is a chart and picture from the web:

No.	String	StringBuffer
1)	String class is immutable.	StringBuffer class is mutable.
2)	String is slow and consumes more memory when you concat too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when you cancat strings.
3)	String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.



The Random class

The Random class is in java.util. So you must include import java.util.*

How to obtain a random integer in the range 0...n

- create a Random object using
  Random rand = *new Random()*
- use *rand.nextInt(n)* to obtain a random integer from 0 to n-1
  // the name *rand* is arbitrary. You can use any name .
- *rand.nextDouble()* returns a double,x, such that 0 ≤ x< 1

For example: to get a random integer in the range 0..100

```
Random r = new Random();
int num = r.nextInt(100);      // notice 0..99
```

To get a random integer in the range 1..100:

```
Random rand = Random();
int value = rand.nextInt(100) + 1;
```

Since rand.nextInt(100) returns a random integer in the range 0..99,
rand.nextInt(100) + 1 give a random integer in the range 1..100

Example: Roll two dice.
The value of each die is a random integer in the range 1..6

```
public int rollDice()
   // rolls two dice and returns the sum of the spots
   {
      Random r = new Random(); // create a Random object
      int die1 = r.nextInt(6) + 1;     // get a random int in the range 1..6
      int die 2 = r.nextInt(6) + 1;
      return die1 + die 2;
   }
```

Another way to get a random int in the range 1..6 is
int n = (int)(6*Math.random() +1);

Using the Random class is a little simple than using Math.random() for random ints

The File class

How to Read from a file:

1. import java.io.*
2. Make a File object using the name of the file on disk:

```
File myFile = new File("something.txt");
//The name myFile is arbitrary. You can use any name.
```

3. Include the lines to be sure the file exists

```
if (!myFile.exists())
{
        System.out.println("File does not exist");
        System.exit(0);
}
```

4. Create a scanner that reads from the file:

```
Scanner input = new Scanner(myFile);
```

5. To read until the end of the file use

```
while(myFile.hasNext())….
```

6. Close the scanner when done i.e. input.close()

7. Include the phrase "throws IOException" after the heading of any methods that use the file for example public static void main(String[] args) **throws IOException**

How to write to a file:

1. import java.io.*
2. Make a File object specifying the name of the output file

```
File outFile = new File("results.txt");
//results.txt is the name of a disk file
// If results.txt does not exist, it will be created, if it is on the disk it will be erased
```

3. Create a PrintWriter object with the File object just created to do the writing PrintWriter

```
pw = new PrintWriter(outFile); pw is an arbitrary name
```

4. Use pw.print and pw.println just as you do with System.out.println(). The only difference is that the output goes to the file and not the screen

5. Close the PrintwriterObject: pw.close()

6. Again you need "**throws IOException**" with the methods headings

Example,

A file (*grades.txt*) contains student names in the form lastname firstname. Each name is followed by three grades.

Read the names and the grades from the file and produce another file, (*average.txt*)with the student name followed by the average of the three grades (as a double)

For example:

Suppose the input file, grades.txt, has data such as
Simpson Homer 40 50 60
Simpson Lisa 90 95 100

Then the output file, averages.txt, would contain
Simpson Homer 50
Simpson Lisa 95

```
import java.io.*;
import java.util.*;
public class StudentGrades
{
  public static void getAverages(String inputFile, String outputFile) throws IOException
  {
     File in = new File(inputFile);  // create a File object
     if (!in.exists())
     {
       System.out.println(inputFile + "does not exist");
       System.exit(0);
     }
     Scanner input = new Scanner (in);  // pass the input file to Scanner

    File out = new File(outputFile);  // create a File object
    PrintWriter pw = new PrintWriter(out);  // create the PrintWriter

    while(input.hasNext())        // while there is more data
    {
       String lastName = input.next();
       String firstName = input.next();
       int sum = 0;
     // read the three grades
     for (int i = 1; i <= 3; i++)
     {
       int grade = input.nextInt();
       sum = sum + grade;
```

```
      }

      double average = sum/3.0;  // notice 3.0 not 3 for real division
      pw.println(lastName+ "  "+firstName+ " "+ average);  // pw writes to the output file
    }
    input.close();
    pw.close();  // if you don't close some output data may be lost

   }

  public static void main(String[] args) throws IOException
  {
    Scanner input = new Scanner (System.in);

    // prompt for the input file name
    System.out.print("Input file name: ");
    String inputFile = input.next();

   // prompt for the output file name
   System.out.print("output file name: ");
    String outputFile = input.next();

   // pass the two file names to the method getAverages
    getAverages(inputFile,outputFile);
  }
}
```

Here is a run of the program.

Running the program **(green is user input)**	**Grades.txt** **(the input file)**	**Averages.txt** **(The output file)**
Input file name: grades.txt **output file name:** averages.txt	Simpson Homer 40 50 60 Simpson Lisa 90 95 100 Simpson Marge 80 70 72 Simpson Bart 42 62 72 Simpson Maggie 90 80 70	Simpson Homer 50.0 Simpson Lisa 95.0 Simpson Marge 74.0 Simpson Bart 58.6666666666 Simpson Maggie 80.0

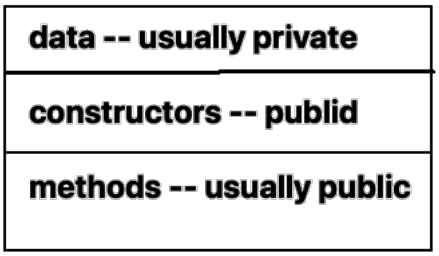
If you want to round the averages to the nearest integer:

long roundedGrade = Math.round(average)
So Bart's average would be 57.

(int)average drops the decimal part. It does not round.

Designing your own classes

The structure of a class:



Private

The data in a class is usually labeled ***private***.

The keyword *private* means that only the methods of the class have direct access to the data.If data is public, any method, either in the class or not can access the data.

Example

```
public class Student
{
        private String name;
        private double gpa;
        .......
}
```

- Private data is hidden inside the the “capsule.”
- The class methods manipulate the data.
- A class method can manipulate the class variables directly. You do not have to pass them as parameters.

Constructors

The “new” operator creates or INSTANTIATES an object

Example:

```
Card c = new Card();
Scanner input = new Scanner(System,in)
String name = new String(“Mary”)
```

When an object is created a special method is automatically called.

This special method is the ***constructor*.**

- The constructor is used to initialize the variable of the class but can do other things as well.
- The name of the constructor is the same as the name of the class.
- There is no return value not even void

Constructors

- The name of the constructor is the name of the class
- A constructor does not have a return value, not even void
- The constructor is called using the word "new"
- The *default constructor* has no parameters or arguments: public Rectangle()
- Other constructors can accept arguments: public Rectangle (int len, int wid)
- If you define no constructors at all then Java provides a default constructor.
- If you define a constructor with arguments then Java will not provide a default constructor. Once you make any constructor, Java stays out of the constructor business.

```
public class Rectangle
{
        private int length, width;
        public Rectangle()    // default constructor
        {
                length = 1;
                width = 1;
        }
        public Rectangle(int len, int wid)    // two- argument constructor
        {
                length = len;
                width = wid;
        }
       // methods of the class such as area() go here
}
```

Create or instantiate two Rectangle objects:

```
public static void main(String[] args)
{
        Rectangle r = new Rectangle(3,4);       //OK
        //This creates or instantiates a  3 x 4 Rectangle object

        Rectangle s = new Rectangle() ;   // call to the default constructor
        //Creates a 1 x 1 Rectangle object
}
```

You can put any statement inside a constructor and when the constructor is called all statements will be executed

```
public Rectangle() // a DEFAULT  constructor with  print statements
{
        length = 1;
        width = 1;
        System.out.println(“I am your default constructor “)
        System.out.println("I set length and width equal to 1");
}
```

Or

```
public Rectangle() // a DEFAULT  constructor that takes input from the
user
{
      Scanner input = new Scanner(System.in);
       System.out.println(“Enter length and width”);
        length = input.nextInt();
       width = input.nextInt();
}
```

```
 Here is a call to the default constructor
public static void main(String[] args)
{
     Rectangle r = new Rectangle();       // calls default constructor
}
```

Output:

```
I am the default constructor
I set length and width equal to 1
```

// No Constructors defined – So Java provides a default constructor

```
public class Rectangle
{
   private int length;
   private int width;


  public int getLength()
  {
    return length;
  }


  public int getWidth()
  {
    return width;
  }

  public static void main(String[] args)
  {
    Rectangle r = new Rectangle();  // calls the Java-supplied default constructor
                                    // length and width are set to 0

    System.out.println("Length is  "+ r.getLength());
    System.out.println("Width is  "+ r.getWidth());
  }
}
```

Output:

```
Length is  0
Width is  0
```

- **Here a one-argument constructor is defined but not a default constructor**
- **Java does not provide a default constructor in this case**
- **So there is no default constructor and a call to a default constructor will be an error**

```
public class Rectangle
{

  private int length;
  private int width;

  public Rectangle(int n)  // One argument constructor
  {
    length = width = n;

  }

  public static void main(String[] args)
  {

   Rectangle2 r = new Rectangle2();       // attempt to call the default – error
   Rectangle2 s = new Rectangle2(5);     // this is OK
  }
}
```

Here is the syntax error message you get from trying to call the non-existent default constructor:

File: C:\ Rectangle2.java [line: 16]
Error: **The constructor Rectangle2() is undefined**

Example

```
 import java.util.*; // for Random class
 public class Dice
 {

  private int numDice;       // how many dice

  // Constructors

  public Dice()  // default constructor
  {
    numDice = 1;
  }

  public Dice (int n)  // one-argument constructor
  {
    numDice= n;
  }

   // other methods

   public int rollDice()
  // rolls numDice dice and returns the sum of the spots
  {
     Random r = new Random();
    int sum = 0;

    for (int i = 1; i <= numDice; i++)
      sum = sum + r.nextInt(6) + 1; // random int 1..6

   return sum;
  }


  public int getNumDice()  // a getter
  {
    return numDice;
  }



  public void setNumDice(int n)  //a setter
  {
    numDice = n;
  }
}
```

```
// A separate class/program that TESTS the Dice class

import java.util.*;

public class TestNumDice
{
   public static void main(String[] args)
   {
     Scanner input = new Scanner(System.in);
     String answer;


     System.out.print("How many dice would you like to toss ?");
     int n = input.nextInt();

      Dice d = new Dice(n); // calls one-argument constructor

     System.out.println("Five rolls of "+ d.getNumDice() +" dice: ");
     for (int i = 1; i <= 5; i++)
              System.out.println(d.rollDice());
    }
}
```