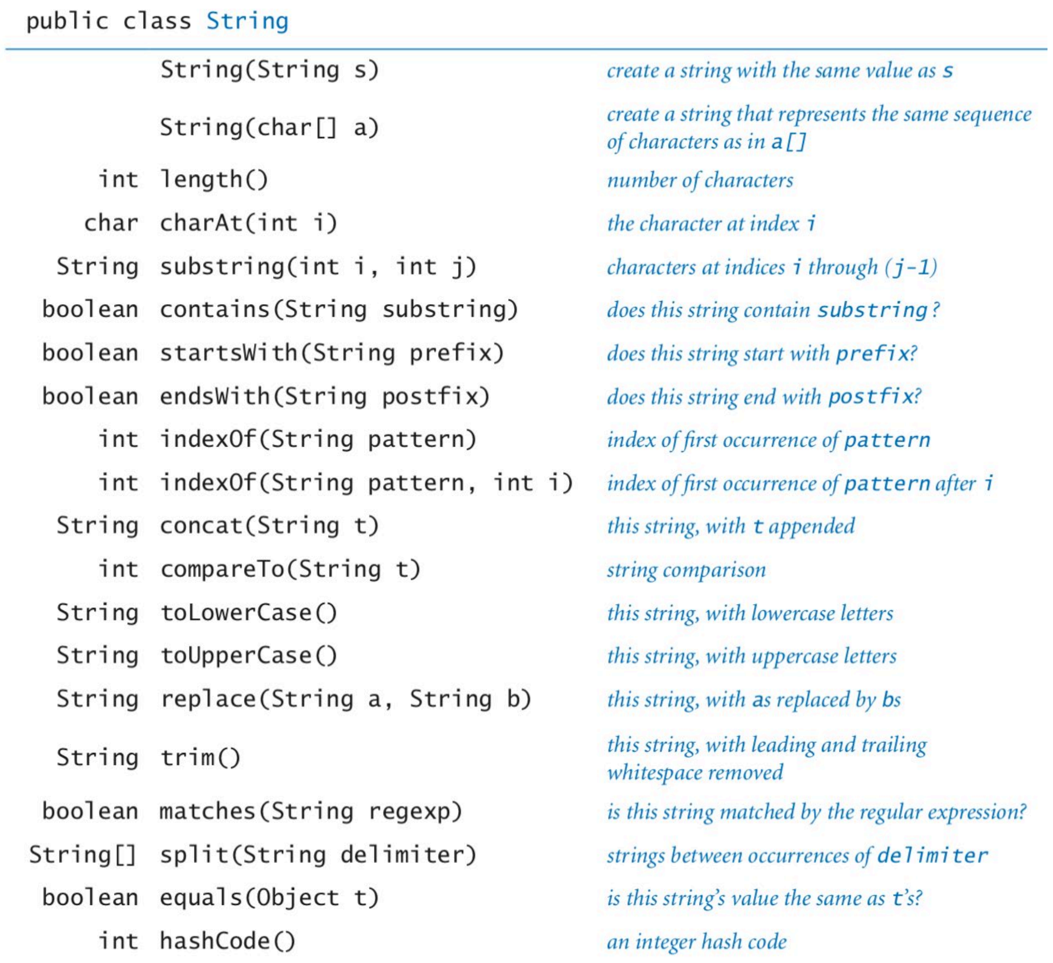
CMSC 113 – COMPUTER SCIENCE 1
Lab#9-Using Data Types/Objects & APIs

In this lab you will practice using Data Types and their APIs.

Task#1: Strings in Java
Recall that the behavior of a data type in an *application programming interface* (API). The **`String`** data type in Java has an API that contains the following (this is a partial list):

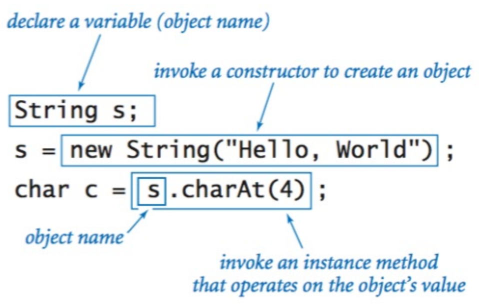


`public class String`

	`String(String s)`	*create a string with the same value as* `s`
	`String(char[] a)`	*create a string that represents the same sequence of characters as in* `a[]`
`int`	`length()`	*number of characters*
`char`	`charAt(int i)`	*the character at index* `i`
`String`	`substring(int i, int j)`	*characters at indices* `i` *through* (`j-1`)
`boolean`	`contains(String substring)`	*does this string contain* `substring`*?*
`boolean`	`startsWith(String prefix)`	*does this string start with* `prefix`*?*
`boolean`	`endsWith(String postfix)`	*does this string end with* `postfix`*?*
`int`	`indexOf(String pattern)`	*index of first occurrence of* `pattern`
`int`	`indexOf(String pattern, int i)`	*index of first occurrence of* `pattern` *after* `i`
`String`	`concat(String t)`	*this string, with* `t` *appended*
`int`	`compareTo(String t)`	*string comparison*
`String`	`toLowerCase()`	*this string, with lowercase letters*
`String`	`toUpperCase()`	*this string, with uppercase letters*
`String`	`replace(String a, String b)`	*this string, with* `a`*s replaced by* `b`*s*
`String`	`trim()`	*this string, with leading and trailing whitespace removed*
`boolean`	`matches(String regexp)`	*is this string matched by the regular expression?*
`String[]`	`split(String delimiter)`	*strings between occurrences of* `delimiter`
`boolean`	`equals(Object t)`	*is this string's value the same as* `t`*'s?*
`int`	`hashCode()`	*an integer hash code*

The first entry, with the same name as the class and no return type, defines a special method known as a *constructor*. The other entries define *instance methods* that can take arguments and return values.

In order to use a data type like **`String`** that is defined as an object type, you must learn how to declare variables, create objects, and then invoke instance methods. Study carefully the example shown below and read the descriptions that follow.



```
String s;
s = new String("Hello, World");
char c = s.charAt(4);
```

declare a variable (object name)
invoke a constructor to create an object
object name
invoke an instance method that operates on the object's value

- ***Declaring variables.*** You declare variables of a reference type in precisely the same way that you declare variables of a primitive type. A declaration statement does not create anything; it just says that we will use the variable name `s` to refer to a **`String`** object.
- ***Creating objects.*** Each data-type value is stored in an *object*. When a client invokes a constructor, the Java system creates (or *instantiates*) an individual object (or *instance*). To invoke a constructor, use the keyword **`new`**; followed by the class name; followed by the constructor's arguments, enclosed in parentheses and separated by commas.
- ***Invoking instance methods.*** The most important difference between a variable of a reference type and a variable of a primitive type is that you can use reference-type variables to invoke the *instance methods* that implement data-type operations (in contrast to the built-in syntax involving operators such as **+*** that we used with primitive types).

Based on the API on Page 1, here are some other string-processing examples.

- ***Data-type operations.*** The following examples illustrate various operations for the `String` data type.

```
String a = new String("now is");
String b = new String("the time");
String c = new String(" the");
```

instance method call	*return type*	*return value*
`a.length()`	`int`	`6`
`a.charAt(4)`	`char`	`'i'`
`a.substring(2, 5)`	`String`	`"w i"`
`b.startsWith("the")`	`boolean`	`true`
`a.indexOf("is")`	`int`	`4`
`a.concat(c)`	`String`	`"now is the"`
`b.replace("t", "T")`	`String`	`"The Time"`
`a.split(" ")`	`String[]`	`{ "now", "is" }`
`b.equals(c)`	`boolean`	`false`

Before proceeding to Task#2, please closely review the material above, including the list of **`String`** methods in the API.

Task#2: Verifying Safe Passwords

In order for passwords to be safe, various services follow safe password practices. These services also have requirements for what qualifies as a safe password. For example, here is a description of safe passwords for a company's online accounts:

Your password must be at least 10 characters long and must contain at least one uppercase letter, lowercase letter, and number and/or symbol.

Write a complete Java program that inputs a user password from the command line, verifies it using the rules above, and declares if the password is valid or not. For example,

```
$ java VerifyPassword “ClaptonIsGod”
<ClaptonIsGod> is not a valid password.

$ java VerifyPassword “Clapton Is God”
<Clapton Is God> is a valid password.
```

Your program (**`VerifyPassword.java`**) should contain the following function:

```
public static boolean verify(String p) {…}
```

that, given a password string p, verifies and returns **`true`**/**`false`** as the case may be. You will use the rules specified above and make use of the **`String`** API to confirm the validity of the password. Write your program in steps, enforcing one rule at a time, and then testing it:

```
// Must contain at least 10 characters
if (p.length() < 10)
   return false;
// Must contains uppercase letters
if …etc.
```

Think carefully about each test and what **`String`** methods might be used. Here are some hints:

- To test if the string contains uppercase letters, you can simply convert the password to all lowercase letters and then compare to see if the resulting string is equal to the original string. Similarly, for checking lowercase letters.
- To test for numbers and symbols, define a new string that contains all the numbers and symbols. Then for each character in the password, if that character is contained in the string of numbers and symbols, the password is valid (only one such character is required by the rules).

Task#3: The Color and Picture APIs
We have seen the Java **`Color`** data type in earlier examples. Here is a more detailed description:

Java's **`Color`** data type represents color values using the RGB color model where a color is defined by three integers (each between 0 and 255) that represent the intensity of the red, green, and blue components of the color. Other color values are obtained by mixing the red, blue and green components. For example, here are some colors and their RGB values:

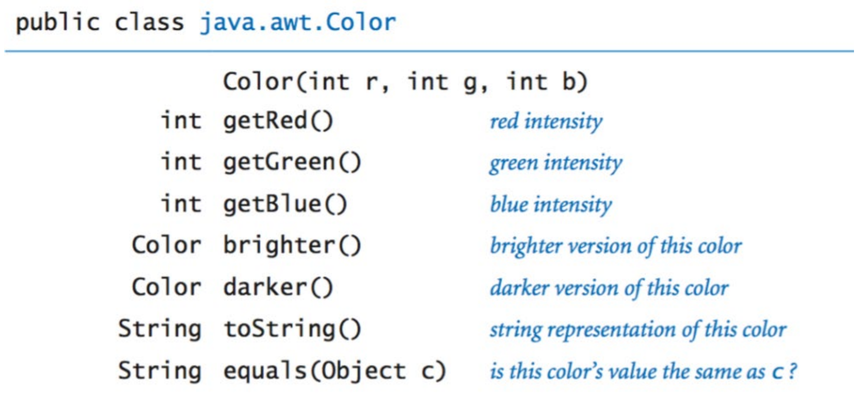
red	*green*	*blue*	
255	0	0	*red*
0	255	0	*green*
0	0	255	*blue*
0	0	0	*black*
100	100	100	*dark gray*
255	255	255	*white*
255	255	0	*yellow*
255	0	255	*magenta*
9	90	166	*this color*

Some color values

The **`Color`** data type has a constructor that takes three integer arguments. For example, you can write

```
Color red      = new Color(255,   0,   0);
Color magenta  = new Color(255,   0, 255);
```

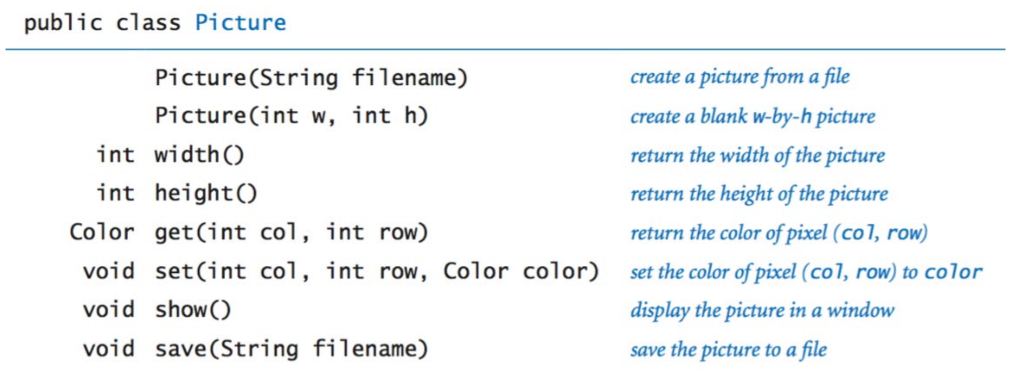
to create objects whose values represent pure red and magenta (see table above). The following table summarizes the methods in the **`Color`** API that we use in this book:



public class java.awt.Color

	`Color(int r, int g, int b)`	
`int`	`getRed()`	*red intensity*
`int`	`getGreen()`	*green intensity*
`int`	`getBlue()`	*blue intensity*
`Color`	`brighter()`	*brighter version of this color*
`Color`	`darker()`	*darker version of this color*
`String`	`toString()`	*string representation of this color*
`String`	`equals(Object c)`	*is this color's value the same as `c`?*

The **`Picture`** class is provided, just like the **`StdOut`**/**`StdIn`** APIs, by the text authors and allows you to load, display, and process images. Here is its API:



public class Picture

	`Picture(String filename)`	*create a picture from a file*
	`Picture(int w, int h)`	*create a blank w-by-h picture*
`int`	`width()`	*return the width of the picture*
`int`	`height()`	*return the height of the picture*
`Color`	`get(int col, int row)`	*return the color of pixel (`col`, `row`)*
`void`	`set(int col, int row, Color color)`	*set the color of pixel (`col`, `row`) to `color`*
`void`	`show()`	*display the picture in a window*
`void`	`save(String filename)`	*save the picture to a file*

First, let us use the **`Picture`** class to see how an image is loaded and displayed:

```
public class Pics {
   public static void main(String[] args) {
      // Get the image file name
      String imageFile = args[0];

      // Load the image
      Picture pic = new Picture(imageFile);

      // Display it
      pic.show();

      // Show data about the image in pic
      StdOut.printf("Image file: %s\n", imageFile);
      StdOut.printf("Width = %d, Height = %d\n", pic.width(), pic.height());

   } // main()
} // Pics
```

Task#4: Enter, compile, and run the program above.
You will need an image file. Some are provided on the class website (where you downloaded this Lab file). Click on the images and then save them in the same folder as the program above. Or create your image files and use them. Here is how you will run the program (on image file: paul-pogba.jpg):

```
$ java Pics paul-pogba.jpg
```

You will see an image of Paul Pogba (member of the French National Soccer team and also of the UK's Premier League club Manchester United) in a window. This is a special, picture window, not the same as the one where you draw your graphics using **`StdDraw`**. Notice how the instance methods **`width()`**, **`height()`**, and **`show()`** are being used above.

Task#5: Learning about *luminance* and *grayscale* of color.
The quality of the images on modern displays such as LCD monitors, plasma TVs, and cell-phone screens depends on an understanding of a color property known as *monochrome luminance*, or effective brightness. It is a linear combination of the three intensities: if a color's red, green and blue values are *r*, *g*, and *b*, respectively then its luminance is defined by the equation

$$Y = 0.299r + 0.587g + 0.114b$$

Grayscale. The RGB color model has the property that when all three color intensities are the same, the resulting color is on a grayscale that ranges from black (all 0s) to white (all 255s). A simple way to convert a color to grayscale is to replace the color with a new one whose red, green, and blue values equal its luminance. See example below:

red	*green*	*blue*		
9	90	166	*this color*	🟦
74	74	74	*grayscale version*	▩
0	0	0	*black*	⬛

0.299 * 9 + 0.587 * 90 + 0.114 * 166 = 74.445

Next, using the **`get()`** and **`set()`** instance methods of **`Picture`** object change all pixels in an image to grayscale. You can use the following function:

```
public static void grayScale(Picture p) {
      // Convert color image p to gray scale image

      int w = p.width();
      int h = p.height();
      for (int col = 0; col < w; col++) {
         for (int row = 0; row < h; row++) {
            Color color = p.get(col, row);
            Color gray = toGray(color);
            p.set(col, row, gray);
         }
      }
   } // grayScale()
```

Given a Picture object **p**, the **`grayscale()`** function converts all pixels of **p** into corresponding grayscale values. You will need to write your own function **`toGray()`** to convert a given color to grayscale using luminance, as described above (you will need to use the **`Color`** data type). Be sure to have the command:

```
import java.awt.Color;
```

At the very top of your program, outside the class definition. This is how you access Java's pre-defined APIs.

Write a complete program that loads a specified image (as in Task#4), converts it into a gray scale image, and displays it.

Task#6: More image processing fun!
If you have time, try the program below on the images posted on the class web page (or you can use your own…selfie!).

```
import java.awt.Color;
public class Obamicon {
   public static void main(String[] args) {
      Color darkBlue = new Color(0, 51, 76);
      Color red = new Color(217, 26, 33);
      Color lightBlue = new Color(112, 150, 158);
      Color yellow = new Color(252, 227, 166);

      String imageFile = args[0];
      Picture pic = new Picture(imageFile);

      StdOut.printf("Press ENTER");

      for (int col = 0; col < pic.width(); col++) {
         for (int row = 0; row < pic.height(); row++) {
            Color color = pic.get(col, row);
            int r = color.getRed();
            int g = color.getGreen();
            int b = color.getBlue();
            int total = r + g + b;
            if (total < 182)
               pic.set(col, row, darkBlue);
            else if (total < 364)
               pic.set(col, row, red);
            else if (total < 546)
               pic.set(col, row, lightBlue);
            else
               pic.set(col, row, yellow);
         }
      }
      pic.show();
   }

} // Obamicon
```