



# CREATING DATA TYPES

Spec

FAQ

Project

Submit

The purpose of this assignment is to give you practice creating data types. The first exercise involves an immutable data type; the second exercise considers a mutable data type.

1. **Color data type.** Write a data type `ColorHSB.java` that represents a color in *hue-saturation-brightness* (HSB) format, along with a sample client. The HSB color format is widely used in color pickers.

HSB = (240°, 100%, 100%)

A color in HSB format is composed of three components:

- The *hue* is an integer between 0 and 359. It represents a pure color on the color wheel, with 0° for red, 120° for green, and 240° for blue.
- The *saturation* is an integer between 0 and 100. It represents the purity of the hue.
- The *brightness* is an integer between 0 and 100. It represents the percentage of white that is mixed with the hue.

Implement the following public API:

```
public class ColorHSB {

    // Creates a color with hue h, saturation s, and brightness b.
    public ColorHSB(int h, int s, int b)

    // Returns a string representation of this color, using the format (h, s, b).
    public String toString()

    // Is this color a shade of gray?
    public boolean isGrayscale()

    // Returns the squared distance between the two colors.
    public int distanceSquaredTo(ColorHSB that)

    // Sample client (see below).
    public static void main(String[] args)

}
```

Here is some more information about the required behavior:

- *Corner cases.* Throw an `IllegalArgumentException` in the constructor if any component is outside its prescribed range (0 to 359 for the hue, 0 to 100 for the saturation and brightness); throw an `IllegalArgumentException` in `distanceSquaredTo()` if its argument is `null`.
- *String representation.* Return a string composed of the integers for hue, saturation, and brightness (in that order), separated by commas, and enclosed in parentheses. An example is `(26, 85, 96)`.
- *Grayscale.* A color in HSB format is a shade of gray if either its saturation or brightness component is 0% (or both).
- *Distance.* The squared distance between two colors  $(h_1, s_1, b_1)$  and  $(h_2, s_2, b_2)$  is defined to be

$$\min \{ (h_1 - h_2)^2, (360 - |h_1 - h_2|)^2 \} + (s_1 - s_2)^2 + (b_1 - b_2)^2$$

For example, the squared distance between (350, 100, 45) and (0, 100, 50) is  $10^2 + 0^2 + 5^2 = 125$ .

- *Sample client.* The `main()` method should take three integer command-line arguments  $h$ ,  $s$ , and  $b$ ;

read a list of pre-defined colors from standard input; and print to standard output the pre-defined color that is closest to  $(h, s, b)$ .

- **Input specification.** The input from standard input consists of a sequence of one or more lines. Each line contains a string (the name of a pre-defined color) and three integers (its hue, saturation, and brightness components), separated by whitespace. The data files [web.txt](#) and [wiki.txt](#) are in the specified format.

% more web.txt				% more wiki.txt			
White	0	0	100	Absolute_Zero	217	100	73
Silver	0	0	75	Acid_Green	65	86	75
Gray	0	0	50	Aero	206	47	91
Black	0	0	0	Aero_Blue	151	21	100
Red	0	100	100	African_Violet	288	31	75
Maroon	0	100	50	Air_Force_Blue_(RAF)	204	45	66
Yellow	60	100	100	Air_Force_Blue_(USAF)	220	100	56
Olive	60	100	50	:			
Lime	120	100	100	Princeton_Orange	26	85	96
Green	120	100	50	:			
Aqua	180	100	100	Yellow_Sunshine	58	100	100
Teal	180	100	50	Zaffre	233	100	66
Blue	240	100	100	Zinnwaldite_Brown	23	82	17
Navy	240	100	50	Zomp	166	66	65
Fuchsia	300	100	100				
Purple	300	100	50				

Diagram annotations: An arrow labeled "data for one pre-defined color" points to the Yellow row. An arrow labeled "name" points to the Blue row. Arrows labeled "hue", "saturation", and "brightness" point to the values 300, 100, and 50 in the Purple row. An arrow labeled "1,296 colors" points to the Zomp row.

- **Output specification.** The output to standard output consists of one line: the name of the nearest pre-defined color and the string representation of that color, separated by whitespace.

```
~/Desktop/oop2> java-introcs ColorHSB 25 84 97 < web.txt
Red (0, 100, 100)

~/Desktop/oop2> java-introcs ColorHSB 350 100 45 < web.txt
Maroon (0, 100, 50)

~/Desktop/oop2> java-introcs ColorHSB 25 84 97 < wiki.txt
Princeton_Orange (26, 85, 96)
```

2. **Clock data type.** Write a data type `Clock.java` that represents time on a 24-hour clock, such as 00:00, 13:30, or 23:59. Time is measured in *hours* (00–23) and *minutes* (00–59). To do so, implement the following public API:

```
public class Clock {

    // Creates a clock whose initial time is h hours and m minutes.
    public Clock(int h, int m)

    // Creates a clock whose initial time is specified as a string, using the format HH:MM.
    public Clock(String s)

    // Returns a string representation of this clock, using the format HH:MM.
    public String toString()

    // Is the time on this clock earlier than the time on that one?
    public boolean isEarlierThan(Clock that)

    // Adds 1 minute to the time on this clock.
    public void tic()

    // Adds Δ minutes to the time on this clock.
    public void toc(int delta)

    // Test client (see below).
    public static void main(String[] args)
}
```

Here is some more information about the required behavior:

- *Two-argument constructor.* Throw an `IllegalArgumentException` if either integer argument is outside its prescribed bounds (hours between 0 and 23, minutes between 0 and 59).
- *One-argument constructor.* The string argument is composed of two digits, followed by a colon, followed by two digits, such as 09:45. Throw an `IllegalArgumentException` if either the string argument is not in this format or if it does not correspond to a valid time between 00:00 and 23:59.
- *String representation.* The format is the hours (2 digits), followed by a colon, followed by the minutes (2 digits). Two examples are 00:00 and 23:59.
- *Ordering.* Times are ordered from 00:00 (earliest) to 23:59 (latest).
- *Tic.* Add one minute to the current time. For example, one minute after 06:00 is 06:01; one minute after 23:59 is 00:00.
- *Toc.* Add  $\Delta$  minutes to the current time. For example, 60 minutes after 12:34 is 13:34. Throw an `IllegalArgumentException` if  $\Delta$  is negative.
- *Test client.* The `main()` method must call each instance method directly and help verify that they work as prescribed.
- *Performance.* All instance methods must take constant time.

**Submission.** Submit a .zip file containing `ColorHSB.java` and `Clock.java`. You may not call library functions except those in the `java.lang` (such as `Integer.parseInt()` and `Math.sqrt()`). Use only Java features that have already been introduced in the course (e.g., objects but not interfaces).

*This assignment was developed by Kevin Wayne.  
Copyright © 2019.*