Java traits

New Austrian Coding School

Programming - handout

String traits

Trait	Description	Example	Output
length()	Returns the length of the String	String name = “Sarah”; Integer length = name.length();	length is 5
isEmpty()	Returns if the String is empty or not	String something = “hello”; String nothing = “”; Boolean isSomethingEmpty = something.isEmpty(); Boolean isNothingEmpty = nothing.isEmpty();	isSomethingEmpty is false isNothingEmpty is true
equals(string)	Returns if the String is equal to *string*, taking into consideration upper and lower cases	String color = “blue”; Boolean answer1 = color.equals(“blue”); Boolean answer2 = color.equals(“Blue”);	answer1 is true answer2 is false
equalsIgnoreCase(string)	Returns if the String is equal to *string*, not taking into consideration upper and lower cases	String color = “blue”; Boolean answer1 = color.equals(“blue”); Boolean answer2 = color.equals(“Blue”);	answer1 is true answer2 is true
startsWith(string)	Returns if the String starts exactly with the given *string*	String greeting = “hello”; Boolean starts = greeting.startsWith(“he”);	starts is true
contains(string)	Returns if the String contains exactly the given *string*	String greeting = “hello”; Boolean contains = greeting.contains(“el”);	contains is true
endsWith(string)	Returns if the String ends exactly with the given *string*	String greeting = “hello”; Boolean ends = greeting.endsWith(“lo”);	ends is true
toUpperCase()	Returns a new copy of the String with all its letters in uppercase	String greeting = “Hello”; String upper = greeting.toUpperCase();	upper is “HELLO”
toLowerCase()	Returns a new copy of the String with all its letters in lowercase	String greeting = “Hello”; String lower = greeting.toLowerCase();	lower is “hello”

trim()	Returns a new copy of the String without spaces at the beginning and at the end	String text = “ hi “; String trimmed = text.trim();	trimmed is “hi”
substring(start, end)	Returns a new String that is a part of the String that starts exactly at *start* and ends exactly at *end* - 1	String text = “Hey there!”; String part = text.substring(1,3);	part is “ey”
replaceAll(target, replacement)	Returns a new copy of String replacing all the occurrences of *target* with *replacement*	String text = “the car is on the street”; String replaced = text.replaceAll(“the”, “that”)	replaced is “that car is on that street”
split(separator)	Returns an Array of String that contains all the pieces that were cut between the given *separator*	String shoppingList = “eggs, tomatoes, paprikas”; String[] ingredients = shoppingList.split(“, “);	ingredients is an Array that looks like [“eggs”, “tomatoes”, “paprikas”]
matches(regex)	Returns if the regular expression given is found within the String. Writing regular expressions is an elaborate process and it is worth reviewing	String quote = "The lips of wisdom are closed, except to the ears of Understanding"; Boolean wisdomMatches = quote.matches(".*wisdom.*"); Boolean wisdomOfDoesNotMatch = quote.matches(".*wisdom of.*"); String a = "a"; String d = "d"; Boolean aMatches = a.matches("[abc]"); Boolean dDoesNotMatch = d.matches("[abc]");	wisdomMatches is true wisdomOfDoesNotMatch is false aMatches is true dDoesNotMatch is false

Math traits

We don’t need to create a Math object. Instead we use the class and the static method.

Trait	Description	Example	Output
Math.ceil(double)	Returns the rounded up version of *double* as a new Double number	double ceil = Math.ceil(3.5);	ceil is 4.0
Math.floor(double)	Returns the rounded down version of *double* as a new Double	double floor = Math.floor(3.5);	floor is 3.0

	number		
Math.abs(number)	Returns the absolute value of *number* as a new Integer	int abs1 = Math.abs(-5); int abs2 = Math.abs(5);	abs1 is 5 abs2 is 5

Randomization

Trait	Description	Example	Output
nextInt()	Returns a random Integer between the negative max Integer and the positive max Integer	int randomInteger = random.nextInt();	randomInteger is 824267639
nextInt(limit)	Returns a random Integer between zero and *limit* - 1	int randomInteger = random.nextInt(5);	randomInteger is 4
nextDouble()	Returns a random Double between 0.0 and 1.0	double randomDouble = random.nextDouble();	randomDouble is 0.5374162143875936
nextBoolean()	Returns either true or false randomly	boolean randomBoolean = random.nextBoolean();	randomBoolean is true

Collection traits

We don't need to create a Collections object. Instead we use the class and the static method.
Assume we use the following list of names: [Lisa, Mona, Sam, Anton]

Trait	Description	Example	Output
Collections.reverse(collection)	Alters *collection* so that its elements are reversed	Collections.reverse(names);	names is [Anton, Sam, Mona, Lisa]
Collections.rotate(collection, shift)	Alters *collection* so that its elements are rotated exactly a *shift* number of steps. If *shift* is positive the elements are rotated to the right. If *shift* is negative the elements are rotated to the left.	Collections.rotate(names, 1); Collections.rotate(names, -1);	names is [Anton, Lisa, Mona, Sam] names is [Mona, Sam, Anton, Lisa]
Collections.sort(collection)	Alters *collection* so that its elements are sorted according to their *compareTo* method	Collections.sort(names);	names is [Anton, Lisa, Mona, Sam]
Collections.shuffle(collection)	Alters *collection* so that its elements are randomly rearranged	Collections.shuffle(names);	names is [Mona, Lisa, Anton, Sam]

Object traits

Because every object **extends** from the class Object, every class we will ever use or create also **has** these traits. If we don't like this default behavior, we can always **overwrite** it in our own class.

Trait	Description	Example	Output
toString()	Returns a new String that represents the String version of that object. Basic types return the right String version that we would expect. But by default, other objects will return the memory address where they are allocated.	Integer number = 5; String numberAsString = number.toString(); Hummus hummus = new Hummus(); String hummusAsString = hummus.toString();	numberAsString is "5" hummusAsString is lectures.week3.traits.Hummus@5e2de80c
equals(other)	Returns if *other* is the same as the original object. The basic types already return the answer we would expect. But by default, other objects will just compare the memory addresses where they are allocated.	Integer number1 = 5; Integer number2 = 2; Boolean result = number1.equals(number2);	result is false
compareTo(other)	Returns zero if *original* is equal to *other*, -1 if it is smaller and 1 if it is bigger. Numbers in general behave like this. Strings in general will arrange alphabetically. Other objects will compare their memory addresses.	Integer number1 = 5; Integer number2 = 2; int comparison = number1.compareTo(number2);	comparison is 1

Changing types

Trait	Description	Example	Output
Using the constructor	Returns a new object that is the right version of the input we provided. This only works in some cases. In more complex cases we will have to take care of the transformation ourselves by providing a proper constructor.	Double decimal = new Double(7);	decimal is 7.0
toString()	Returns the String version of the object. Basic types already return what we would expect. But other objects will just return their memory address.	Integer number = 7; String seven = number.toString();	seven is “7”
valueOf(other)	Returns a new object that is the right version of the input we provided. This only works in some cases.	Integer fiftySeven = Integer.valueOf("57");	fiftySeven is 57