Recitation 4 - January 30th A4/GR

William and Isaac

Methods:

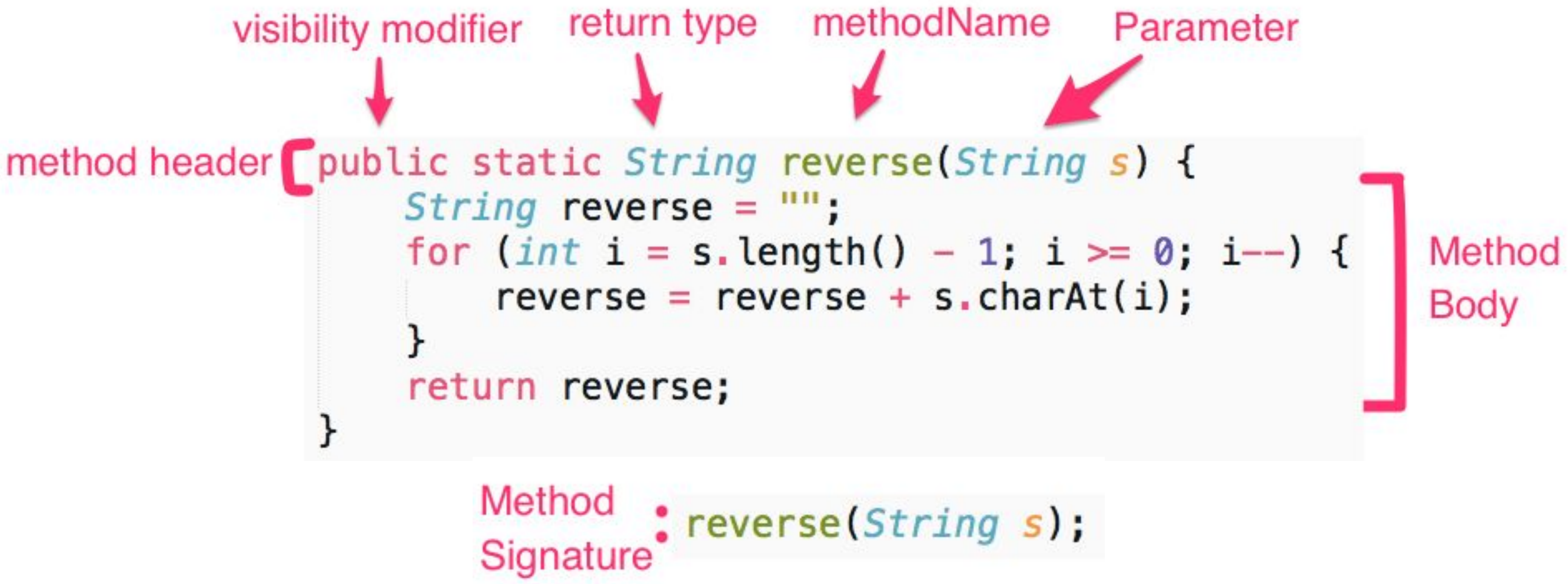
All methods have the same general layout:

```
public static return_type method_name(parameter_list) {
    //method_body
}
```

And they have the following components:

- Method Name
- Parameter List
- Return Type: only one thing can be returned (may be an Object) and can be void.
- Method Signature: the method name and parameter list
- Method Header: whole first line
- Method Body: inside curly braces is your main body of code, includes your return statement.

Some Definitions:



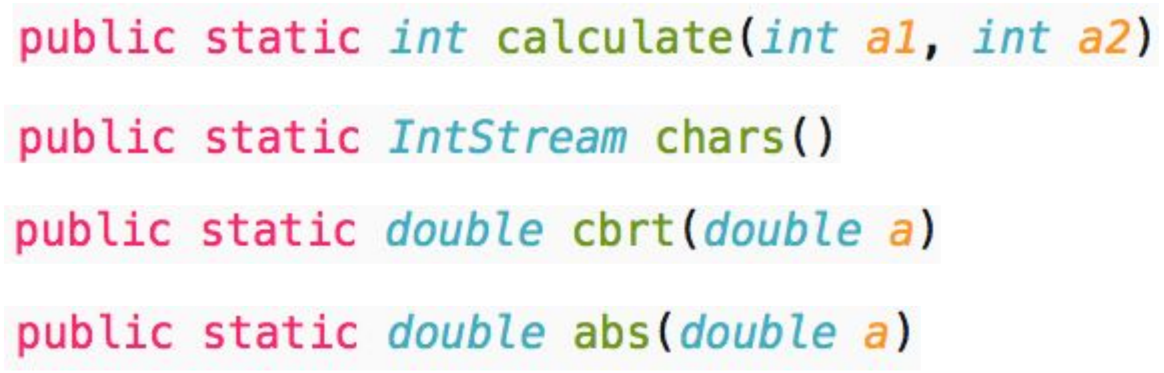
```
public static String reverse(String s) {
    String reverse = "";
    for (int i = s.length() - 1; i >= 0; i--) {
        reverse = reverse + s.charAt(i);
    }
    return reverse;
}
```

```
reverse(String s);
```

Methods end when a return statement is called.

Describe the Method

From the following methods, tell me their return type, method name, and parameter name.



```
public static int calculate(int a1, int a2)

public static IntStream chars()

public static double cbrt(double a)

public static double abs(double a)
```

Pass by value

Java methods are pass by value: a parameter's value is copied into a local variable (formal parameter) in method. (this is as opposed to pass by reference, where you could change the value of the variable from within the method)

Arrays

- Arrays are **objects** that hold a fixed number of things (primitive or object types)
    - Once created, an array's length cannot change
- Arrays are ordered; each element (item) has an index (position)
    - Like any sane programming language, indexing starts at 0
    - What's the last index?
- When an array is created, every element is set to the type's default value (0/null)

Creating Arrays

```
int[] nums;

int nums[];
```

- While these both declare an array of integers, the first is preferred

```
nums = new int[10];

double[] arr = new double[5];

String[] str = {"one","two"};
```

Creating Arrays

Multidimensional!

```
int[][] mat = new int[3][3];
```

- An array of length 3 with each element being an array of length 3, i.e. a 3x3 array/matrix

```
int[][] arr = new int[3][];
```

- Can have jagged arrays by individually creating the sub-arrays, but they're not quite as useful

Using Arrays

```
arr.length
```

- Gives you the length of an array

```
String s = str[1];
```

- s will contain...?

```
str[0] = "zero";
```

- Sets the 0th element to "zero"

```
String fun = str[3];

str[2] = ":)";
```

For-each Loops

The easy for loop

```
int[] evens = {2,4,6,8,10};

for (int num : evens) {

    System.out.println(num);

}
```

- What will print?

For-each Loops

The easy for loop

```
2

4

6

8

10
```

Import Statements

- By default, Java is only able to see classes in the java.lang package
- To gain access to more classes, enums, etc., we put import statements at the beginning of the file
- `import java.util.Random;`
  - Imports the `Random` class from the package `java.util`
- `import java.util.*;`
  - Imports everything from the package `java.util`
  - Don't do this, bad practice/checkstyle error

Random

- Class in `java.util` used to generate random numbers
- Before getting numbers, we must create an instance of `Random` using a constructor
- Calling methods on the object will give us random numbers

Creating an Instance of Random

```
Random rand = new Random();
```

- Creates a random number generator

```
Random randS = new Random(1);
```

- Creates a "seeded" random number generator

Using Random

Getting `int` values

```
int a = rand.nextInt();
```

- Generates a random integer value

```
int b = rand.nextInt(bound);
```

- Generates a random integer 0-(bound - 1), i.e. [0, bound)

Using Random

Controlling `nextInt`'s range

```
int c = a + rand.nextInt(b);
```

- c is an int from a - (a + b - 1)

```
age = 18 + rand.nextInt(6);
```

- x is a number 18 - 23

```
num = 1 + rand.nextInt(6);
```

Using Random

Getting `double` values

```
double d = rand.nextDouble();
```

- Generates a decimal value [0.0,1.0), i.e. never get 1.0

Using Random

Getting `boolean` values

```
rand.nextBoolean();
```

- Generates a true or false, no catch here

Math

- Available by default (in `java.lang`)
- Contains some useful math-related methods
- No need to create an instance of it like `Random`, as almost all methods are static

Math Methods

"Rounding" methods

```
Math.ceil(num);
```

- "ceiling," returns smallest int greater than num (always rounds up)

```
Math.floor(num);
```

- Opposite of ceil, returns greatest int smaller than num (always rounds down)

Math Methods

Actual rounding method

```
Math.round(num);
```

- Rounds like you think it should.

NOTE:

- double -> long
- float -> int

Math Methods

Works with pretty much any type

```
Math.max(a, b);
```

- Returns a or b, whichever is larger

```
Math.min(a, b);
```

- Returns a or b, whichever is smaller

Enums!

Aka enumerable types

- Is a special data type that enables for a variable to be a set of predefined constants.
- The variable must be equal to one of the values that have been predefined for it.

Creating Enumerated Types

```
public enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
}

//Notice that enums are constants! So they should be defined with all
capital letters
```

Defining Enums and Assigning them a value:

- We can call the value of an enum by writing:
  - `enumName.CONSTANTVALUE`
  - `Ex: Day.TUESDAY`

- And we can create a new Day Enum by:
  - `Day day;`

- So we can assign it a value by:
  - `day = Day.TUESDAY;`

Enum Methods!

.ordinal(): Returns the ordinal of this enumeration constant (its position in its enum declaration, where the initial constant is assigned an ordinal of zero).

- Ex: `day.ordinal() = 3; //Since day = Day.WEDNESDAY`

.values(): returns a list of all values that the Enum could be:

- Ex: `Day.values() = {SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY}`