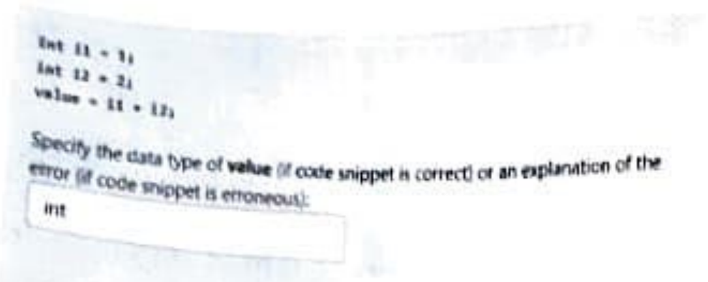
1. Task: Simple Code Snippets

In the Moodle module associated with this task, you can find a number of smaller co snippets. In each of these code snippets is a variable **value** for which no data type is specified. Your task (if the code is error-free) is to specify the appropriate data type. If there is an error in the code, briefly state what the error is.

Example:

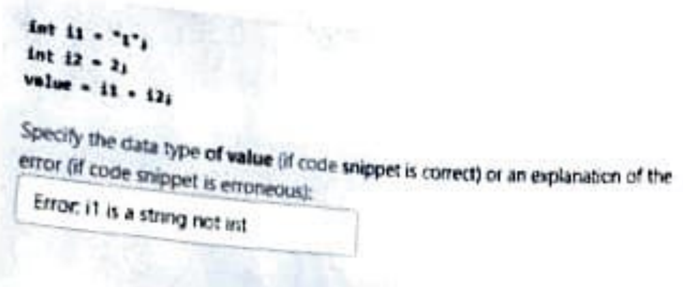


```
int i1 = 1;
int i2 = 2;
value = i1 + i2;
```

Specify the data type of **value** (if code snippet is correct) or an explanation of the error (if code snippet is erroneous):

int

Abbildung 1: Example of a valid code snippets



```
int i1 = "1";
int i2 = 2;
value = i1 + i2;
```

Specify the data type **of value** (if code snippet is correct) or an explanation of the error (if code snippet is erroneous):

Error: i1 is a string not int

Abbildung 2: Example of an erroneous code snippets with explanation

2. Task: Understanding Source Code

Consider the following method **f**:

```
public static int[] f(int[] a, int[] b){
    if(a.length != b.length) return null;

    int[] c = new int[a.length];
    for (int i = 0; i < a.length; i++) {
        c[i] = a[i] * b[i];
    }

    return c;
```

(a) Explain in words what the method **f** calculates and returns. Do not describe each line separately; two or three properly formulated sentences should be enough! (8 P)

(b) Implement the algorithm from `f` in a method `g`. This new method must provide the same results as `f` for all parameter values. However, `g` is *not allowed* to use a `for` statement. Of course, calling `f` is *not allowed* either. (7 P)

The source code for (a) and a suitable `main` method are available in Moodle under `Task 02b`. Your solution to (a) should be written in the corresponding Moodle module (`Task 02a`), the solution to (b) should be stored in the file `Task.java` at `Task 02b` in Moodle; an evaluation with simple examples is provided there.

3. Task: Strings I (10 P)

Anagrams are words, which, by interchanging the letters, result in a new word or a new sentence. The order of the letters does not matter. Thus, new words are formed from letters that actually originate from other words. Here not only parts of other words are used, but always all existing letters. For example, "this" is an anagram of "hist" or "hits"; "cat" is an anagram of "act".

Implement the method

```
boolean isAnagram(String s1, String s2)
```

which is given two strings `s1` and `s2`. The purpose of the method is to check whether `s2` is an anagram of `s1`. If it is, the method shall return `true`, otherwise `false`. **ATTENTION:** You may assume that all words are always lowercase, i.e. instead of "Cat" the word "cat" is passed. Also, you can assume that no letters are duplicated, such as in the word "mom".

Example:

```
s1: this
s2: hits
The strings s1 and s2 are anagrams
```

A general scheme for the solution can be found in Moodle under `Task 03` in the file `Task.java`, with the possibility of a test evaluation by `Evaluation.java`. The solution must be saved in the file `Task.java`.

4. Task: Strings II (15 P)

This task is about processing strings. Implement a method

```
String[] divide(String str, int parts)
```

with two input parameters. The first input parameter (`String str`) is a string containing a sentence or sequence of letters. The second parameter (`int parts`) is a number that specifies how many equal parts `str` should be divided into. You may assume that `str` can always be divided into equal parts, i.e. `str.length() % parts == 0`. Furthermore, the method shall output a `string` array containing all equal-sized parts of the string `str`. It is important that the returned `String` array is only as large as needed and does not contain any empty fields.

Examples:

```
Enter your Sentence: abcdefghijklmnopqrst
Number of Parts: 5
```

```
Divided String:
abcd
efgh
ijkl
mnop
qrst
```

A general scheme for the solution can be found in Moodle under **Task 04** in the file **Task.java**, with the possibility of a test evaluation by **Evaluation.java**. The solution must be saved in the file **Task.java**.

5. Task: Recursion (15 P)

Implement the following method **countSubstring(String str, String substr)** as a recursive method with return type **int** and two **String** input parameters. The goal of this method is to count how many times the substring **substr** is contained in the string **str**. You may assume that both **str** and **substr** are never **null**.

?

Examples:

- `countSubstring("catcatdogcatcat","cat") = 4`
- `contains("catcatdogcat","cat") = 3`
- `contains("catcatdogcat","dog") = 1`

① new method
4
if
return

Furthermore, your solution is not allowed to use any variables other than the parameters and possibly local variables, loops are also not allowed. (Otherwise your solution will be evaluated with 0 points, even if it gives the correct results. This applies analogously to solutions that only reproduce the above example results.)

A general scheme for the solution can be found in Moodle under **Task 05** in the file **Task.java**, with the possibility of a test evaluation by **Evaluation.java**. The solution has to be saved in the file **Task.java**.

6. Task: Arrays (12 P)

In the given class **Evaluation** two **int** arrays **int[] a** and **int[] b** are read. Implement the method

```
int[] intersection(int[]a, int[] b)
```

which takes the two **int** arrays as input parameters. The goal of the method is to determine the intersection of both arrays and output it in a new **int** array. You can assume that the read arrays behave like sets and each number is contained only once. Also note that the array created is to be no larger than the intersection.

Example:

$$intersection([1\ \ 2\ \ 3],[2\ \ 3\ \ 4]) = [2\ \ 3]$$

A general scheme for the solution can be found in Moodle under **Task 06** in the file **Task.java**, with the possibility of test evaluation by **Evaluation.java**. The solution must be saved in the file **Task.java**.

7. Task: Object Oriented Programming Concepts (10 P)

The following task consists of three subtasks to assess basic knowledge in the area of object-oriented programming. Given is a class **Evaluation** which can be used to test your solution. **Attention:** The class and instance variables of the classes you implement should all be externally accessible. Implement the following classes:

a) Implement the class **Person**, which stores a name (`String name`) and an automatically incrementing ID (`int id`). The ID should start at 1000. Also implement a corresponding constructor for the class, which only gets as input parameter the name of the person.

b) Implement a class `NumberList` so that it stores a list (i.e. `LinkedList<> list`) as an instance variable. This list is used to store `int` numbers. The constructor of the `NumberList` class is only supposed to initialize the list. They also implement a method `void addNumber(int number)` which has to add the passed number (`number`) to the list (`list`).

c) In the `Calculator` class, implement a method `int sum(int[] numbers)` that calculates the sum of all numbers stored in `numbers`. Next, check in the method if any of the numbers in `numbers` are negative. If so, your method should throw an `IllegalArgumentException` with the message "Only nonnegative integers".

A general scheme for the solution can be found in Moodle under `Task 07`, with the possibility of an evaluation by `Evaluation.java`. The solution must be saved in the files `Person.java`, `NumberList` and `Calculator`

8. Task: Collections (15 P)

Your task is to implement a trivial social media system. In this task, the system to be developed consists of two components: the `SocialMediaSystem` class and the `Post` class.

The `Post` class is given and stores the content of a post (`String content`) and a list of all hash tags used (`List<String> hashTags`). The `Post` class has a suitable constructor, a `toString()` method and all the important getter methods.

Implement the `SocialMediaSystem` class. An object of this class stores a list of all published posts (`List<Post> posts`). Also implement the following aspects:

- A corresponding constructor that initializes the list of posts (`posts`).
- A method `void post(String content, String hashTags)` which creates a `Post` object from the two strings. To do this, you must first create a `List<String>` from the string `hashTags`. The passed string `hashTags` always has a form like this:

  #computer #science #data #mining

  You can further assume that a blank space is always inserted between the individual hash tags. After you have created a `post` object from the two strings `content` and `hashTags`, you should add it to the list of posts (`posts`).

- Next, implement the `List<Post> collect(String keyword)` method, which collects all posts from `posts` that contain the passed keyword `keyword` either in the post's content (`content`) or in the list of hash tags (`hashTags`). Note that partial matches in the list of posts must also be considered. For example, the following list of hashtags includes the word "good":

#work #monday #goodluck

- Finally, implement the **void printStatistics()** method. The goal of this method is to output statistics about the used hash tags in the following form:

```
#tesla: 1
#spacex: 1
#computer: 2
#science: 2
#turing: 1
#data: 1
#ai: 1
#machine: 1
#learning: 1
```

  Note that there is no evaluation for the last method to check. You can output the hash tags in any order.

A general scheme for the solution can be found in Moodle under `Task 08` with the possibility of an evaluation by `Evaluation.java`. The solution must be saved in the files `Post.java` and `SocialMediaSystem.java`.

9. Task: Inheritance (15 P

In this task, your goal is to implement the individual components of a bus ticketing system.

First, implement the class **BusTicket**. It should store as instance variables an automatically incrementing ID (`int id`), a purchase date (`String date`) and the ticket price (`double price`). Implement a constructor of the following form

```
public BusTicket(String date, double price)
```

and all getter methods.

Then derive the **AnnualTicket** class from the **BusTicket** class. An object of the **AnnualTicket** class additionally stores the name of the person (`String name`) who bought the ticket. Implement a constructor for the class of the following form

```
public AnnualTicket(String date, double price, String name)
```

as well as all required getter methods.

Next, derive the class `FourWayTicket` from the class `BusTicket`. An object of the class `FourWayTicket` should store how many trips are remaining with the ticket. Implement a constructor of the following form

```
public FourWayTicket(String date, double price)
```

For this class, you don't need to implement any additional getter methods.

Next, realize an interface `Expirable` with the following methods:

- `void useTicket()`
- `int getRidesLeft()`