|  |  |  |  |
| --- | --- | --- | --- |
| **Java revisões** | | | |
| 1 | Please type in a code to declare two variables of type int and print their sum.  int x = 4;  \_\_\_\_ y = 7;  int sum = x \_\_\_ y;  System.out.println(\_\_\_ ); |  |  |
|  |  |  |  |
| 2 | In every Java program...  A...there must be at least two variables declared.  B...there must be a method called "main".  C...all of the variables must be integers. |  |  |
|  |  |  |  |
| 3 | Fill in the blanks to output the name:  \_\_\_\_\_\_\_  name; name = "David";  \_\_\_\_\_\_\_ .out.println(\_\_\_\_\_\_\_\_);  String System Java name int |  |  |
|  |  |  |  |
| 4 | Fill in the blanks to print "You rock!" if variable "a" is greater than 15, and variable "b" is less than or equal to 72.  int a = 144;  int b = 33;  if (a > 15 \_\_ b <= \_\_ ) {  System.out.println("You rock!");  } |  |  |
|  |  |  |  |
| 5 | Fill in the blanks to print "in a loop" 7 times, using the while loop.  int x = 1;  while (x <= \_\_\_) {  System.out.println("in a loop");  \_\_\_++;  } |  |  |
|  |  |  |  |
| 6 | Please select the correct statements about && and || operators.  Select all that apply  a) a && b is false if both a and b are true  b) (a || b ) && c is true if c is true and either a or b is true  c) a && b is true if either a or b is true  d) a || b is true if either a or b is true  **6.1. Cria uma tabela com os operadores relacionais e lógicos utilizados na linguagem JAVA** |  |  |
| 7 | What is the output of this code?  int arr[ ] = new int[3];  for (int i = 0; i < 3; i++) {  arr[i] = i;  }  int res = arr[0] + arr[2];  System.out.println(res); |  |  |
| 8 | Fill in the blanks to print "in a loop" 5 times using the for loop.  \_\_\_ (int x = 0; \_\_\_< 5; x++) {  System.out.println("in a loop"); |  |  |
| 9 | What is the output of this code?  int result = 0; for (int i = 0; i < 5; i++) { if (i == 3) {  result += 10; } else { result += i; }  } System.out.println(result); |  |  |

|  |  |  |
| --- | --- | --- |
| Classes and Objects Object-Oriented Programming | | |
|  |  | **POO (programação orientada a objetos) é uma ideia que vê o objeto como uma parte de um programa (como se fossem vários mini programas que juntos fazem um programa).**  **Class – é planta (o projeto da casa mas não é a casa em si)**  **A class é a ideia e o objeto é a coisa em si Conceito1: Podemos fazer várias casas (objetos) com base numa planta (class)  Conceito2: Não serve de nada criar uma planta (class) se não se vai fazer uma casa(objeto)**  **A class define os Atribute e Behaviour – Se se trata-se de uma pessoa (Atributos – Sexo; idade; altura) (Behavior – correr; andar; saltar)** |
|  | **Pergunta C1**  A class defines... (choose two)  Select all that apply   1. values 2. objects 3. attributes 4. behavior | **Object-Orientation**  Java uses **O**bject-**O**riented **P**rogramming (OOP), a programming style that is intended to make thinking about programming closer to thinking about the real world. In OOP, each object is an independent unit with a **unique identity**, just as objects in the real world are. An apple is an object; so is a mug. Each has its unique **identity**. It's possible to have two mugs that look identical, but they are still separate, unique objects. Objects also have **characteristics**, which are used to describe them.  For example, a car can be red or blue, a mug can be full or empty, and so on. These characteristics are also called **attributes**. An attribute describes the current state of an object. In the real world, each object behaves in its own way. The car moves, the phone rings, and so on. The same applies to objects: **behavior** is specific to the object's type. In summary, in object oriented programming, each object has three dimensions: **identity**, **attributes**, and **behavior**.  Attributes describe the object's current state, and what the object is capable of doing is demonstrated through the object's behavior.  **Classes**  A **class** describes what the object will be, but is separate from the object itself.  In other words, classes can be described as blueprints, descriptions, or definitions for an object. You can use the same class as a blueprint for creating multiple objects. The first step is to define the class, which then becomes a blueprint for object creation.  Each class has a name, and each is used to define **attributes** and **behavior**. Some examples of attributes and behavior:  https://api.sololearn.com/DownloadFile?id=2429 |

|  |  |  |
| --- | --- | --- |
|  | Classes and Objects  Methods | |
|  | **Pergunta M1**    Fill in the blanks to call the method "hello" from main:  public static void main(String[ ] args) {  \_\_\_\_\_\_\_\_\_\_\_\_\_\_;  }  static void hello() {  System.out.println("hi");  } | **Methods**  Methods define **behavior**. A method is a collection of statements that are grouped together to perform an operation. System.out.println() is an example of a method.  You can define your own methods to perform your desired tasks.  Let's consider the following code:  class MyClass {  package aulascon;  class Minhaclass {  static void dizola() {  System.out.println("OLÁ mundo!");  }  public static void main(String[ ] args) {  dizola();    }  }  The code above declares a method called "sayHello", which prints a text, and then gets called in **main**. To call a method, type its name and then follow the name with a set of parentheses. |
|  | **Pergunta M2**  How many times can you call a method?   1. one 2. only two 3. as many as you want | **Calling Methods**  You can call a method as many times as necessary.  When a method runs, the code jumps down to where the method is defined, executes the code inside of it, then goes back and proceeds to the next line. **Example:**  package aulascon;  class Minhaclass {  static void dizola() { //Criação do Método dizola  System.out.println("OLÁ mundo!"); //Criação do Método dizola  }  public static void main(String[ ] args) {  dizola(); //Chamada do Método dizola  dizola(); //Chamada do Método dizola  dizola();  dizola();  dizola();    }  } |
|  | **Pergunta M3**  What output results from this code?  public static void main(String[ ] args) {  doSomething(4);  }  static void doSomething(int x) {  System.out.println(x\*x);  } | **Method Parameters**  You can also create a method that takes some data, called **parameters**, along with it when you call it. Write parameters within the method's parentheses. For example, we can modify our **sayHello**() method to take and output a **String** parameter.  class MyClass {  package aulascon;  class Minhaclass {  static void dizola(String nome) {  System.out.println("OLÁ mundo!"+nome);  }  public static void main(String[ ] args) {  dizola("Rui");  dizola("Ana");    }  }  The method above takes a String called **name** as a parameter, which is used in the method's body. Then, when calling the method, we pass the parameter's value inside the parentheses. Methods can take multiple, comma-separated parameters. The advantages of using methods instead of simple statements include the following: - **code reuse**: You can write a method once, and use it multiple times, without having to rewrite the code each time. - **parameters**: Based on the parameters passed in, methods can perform various actions. |