|  |  |
| --- | --- |
|  | Research the Git workflow and Java data types. |
|  | Write about what you learned in your research as though you are teaching someone else the concepts. |
|  | Provide 5 sources used in your research.  <https://www.youtube.com/watch?v=6LhTe8Mz6jM>  <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>  Gitflow Workflow is a Git workflow design that was first published and made popular by [Vincent Driessen at nvie](http://nvie.com/posts/a-successful-git-branching-model/). The Gitflow Workflow defines a strict branching model designed around the project release. This provides a robust framework for managing larger projects.  **What is Git flow?**   1. A set of guidelines developers can follow when using version control. 2. Referred to as a “Branching Model” 3. Not rules that are set in stone, but guidelines   **How does it work?**   1. Central Repository 2. Your team of Developers will clone central repository and work locally and from there push branches 3. There are two branches used to record project history, Master and Develop 4. Develop serves as an integration branch for features 5. Master branch stores the official release history   **Creating a feature**   1. You need to start working on a new feature 2. Pull latest copy of “develop” 3. Fork “develop” and create own “feature branch 4. When code is completed and tested we will merge “feature branch” into “develop”   **Release Branches**   1. Release branches are created by forking “develop.” 2. Senior developer will create a “release branch” 3. Release branch will contain a determined amount of features 4. Release branch should be deployed to a staging server to QA testing 5. Any bugs, needs to be addressed on the release branch 6. The release branch will have to be merged back into “develop” as well as “master.” 7. You should then tag “master” with a version number.   Managing Hotfixes   1. Hotfixes are defined as minor fixes to the project 2. Fork “master” to create a new “hotfix” branch. 3. Commit code to the “hotfix” branch. 4. The “hotfix” branch, once tested, must be merged into the “master” and “develop.” 5. The “master” branch should be tagged again and deployed.   Java Data Types  <https://www.geeksforgeeks.org/data-types-in-java/>  <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>  <https://chortle.ccsu.edu/java5/Notes/chap09C/ch09C_2.html>  <https://www.sitepoint.com/beginning-java-data-types-variables-and-arrays/>  Java has two categories of data:   1. Primitive data (8 types) 2. Object data (programmer created types) Non-primitive—which include Classes, Interfaces, and Arrays.   Primitive Data Types:  There are only eight primitive data types in Java: byte, short, int, long, float, double, char, and boolean.  **byte**: The byte data type is an 8-bit signed two's complement integer.It has a minimum value of -128 and a maximum value of 127 (inclusive). The byte data type can be useful for saving memory in large [arrays](https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html), where the memory savings actually matters. They can also be used in place of int where their limits help to clarify your code; the fact that a variable's range is limited can serve as a form of documentation.  **short**: The short data type is a 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive). As with byte, the same guidelines apply: you can use a short to save memory in large arrays, in situations where the memory savings actually matters.  **int**: By default, the int data type is a 32-bit signed two's complement integer, which has a minimum value of -231 and a maximum value of 231-1. In Java SE 8 and later, you can use the int data type to represent an unsigned 32-bit integer, which has a minimum value of 0 and a maximum value of 232-1. Use the Integer class to use int data type as an unsigned integer. See the section The Number Classes for more information. Static methods like compareUnsigned, divideUnsigned etc have been added to the [Integer](https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html) class to support the arithmetic operations for unsigned integers.  **long**: The long data type is a 64-bit two's complement integer. The signed long has a minimum value of -263 and a maximum value of 263-1. In Java SE 8 and later, you can use the long data type to represent an unsigned 64-bit long, which has a minimum value of 0 and a maximum value of 264-1. Use this data type when you need a range of values wider than those provided by int. The [Long](https://docs.oracle.com/javase/8/docs/api/java/lang/Long.html) class also contains methods like compareUnsigned, divideUnsigned etc to support arithmetic operations for unsigned long.  **float**: The float data type is a single-precision 32-bit IEEE 754 floating point. Its range of values is beyond the scope of this discussion, but is specified in the[Floating-Point Types, Formats, and Values](https://docs.oracle.com/javase/specs/jls/se7/html/jls-4.html#jls-4.2.3) section of the Java Language Specification. As with the recommendations for byte and short, use a float (instead of double) if you need to save memory in large arrays of floating point numbers. This data type should never be used for precise values, such as currency. For that, you will need to use the [java.math.BigDecimal](https://docs.oracle.com/javase/8/docs/api/java/math/BigDecimal.html) class instead. [Numbers and Strings](https://docs.oracle.com/javase/tutorial/java/data/index.html) covers BigDecimal and other useful classes provided by the Java platform.  **double**: The double data type is a double-precision 64-bit IEEE 754 floating point. Its range of values is beyond the scope of this discussion, but is specified in the [Floating-Point Types, Formats, and Values](https://docs.oracle.com/javase/specs/jls/se7/html/jls-4.html#jls-4.2.3) section of the Java Language Specification. For decimal values, this data type is generally the default choice. As mentioned above, this data type should never be used for precise values, such as currency.  **boolean**: The boolean data type has only two possible values: true and false. Use this data type for simple flags that track true/false conditions. This data type represents one bit of information, but its "size" isn't something that's precisely defined.  **char**: The char data type is a single 16-bit Unicode character. |
|  |  |