# **ACTIVITY:11**

* Study DRY principle, Identify the benefits:

DRY is simply an approach, or we can say, a different perspective to programmers. DRY stands for **Don’t Repeat Yourself**. In Java, it means don’t write the same code repeatedly. Suppose you are having the same code at many places in your program, then it means you are not following the DRY approach; You are repeating the same code at different places. Hence, the solution is obtained using the DRY concept by placing the methods in place of all repeated codes and defining the code in one method. So by calling methods, we will reach the principle DRY. The DRY concept is very important to make the code better by reducing code redundancy and to encourage its reusability.

**Applications:**

* Online marketing applications
* Education
* Financial applications

**Illustration 1:**

*Consider the scenario of the college student system. The college contains many departments. So each department has different people, but the college name is the same. So no need to specify the college name for each department by writing the code for the display of the college name.*

**Implementation:** Without DRY approach

**Example 1:**

## Java

|  |
| --- |
| // Java Program without DRY approach    // Main class  **public** **class** GFG {        // Method 1      // For cse department  **public** **void** CSE()      {          System.out.println("This is computer science");      }        // Method 2      // For cse dept. college  **public** **void** college()      {          System.out.println("IIT - Madras");      }      // Method 3      // ece dept method  **public** **void** ECE()      {          System.out.println("This is electronics");      }        // Method 4      // For ece dept college 1  **public** **void** college1()      {          System.out.println("IIT - Madras");      }      // Method 5      // For IT dept  **public** **void** IT()      {          System.out.println(              "This is Information Technology");      }        // Method 6      // For IT dept college 2  **public** **void** college2()      {          System.out.println("IIT - Madras");      }        // Method 7      // Main driver method  **public** **static** **void** main(String[] args)      {            // Creating object of class in main() method          GFG s = **new** GFG();            // Calling above methods one by one          // as created above          s.CSE();          s.college();          s.ECE();          s.college1();          s.IT();          s.college2();      }  } |

**Output**

This is computer science

IIT - Madras

This is electronics

IIT - Madras

This is Information Technology

IIT - Madras

**Implementation:** Applying the DRY principle

* Here we create only one method named college and then call the method in all the departments.

**Example 1:**

## Java

|  |
| --- |
| // Java Program with Use of DRY Concept    // Importing input output classes  **import** java.util.\*;    // Main class  **public** **class** GFG {        // Method 1      // For cse department  **public** **void** CSE()      {            // Print statement          System.out.println("This is computer science");            // Calling method          college();      }        // Method 2      // For ece dept method  **public** **void** ECE()      {          System.out.println("This is electronics");            // Calling method          college();      }        // Method 3      // For IT dept  **public** **void** IT()      {            // Print statement          System.out.println(              "This is Information Technology");            // Calling method          college();      }        // Method 4      // For college dept  **public** **void** college()      {            // Print statement          System.out.println("IIT - Madras");      }        // Method 5      // Main driver method  **public** **static** **void** main(String[] args)      {            // Creating object of class in main() method          GFG s = **new** GFG();            // Calling the methods one by one          // as created above          s.CSE();          s.ECE();          s.IT();      }  } |

**Output**

This is computer science

IIT - Madras

This is electronics

IIT - Madras

This is Information Technology

IIT - Madras

DRY – Simple example

Assume you have many places in your code that need to be executed based on the current user’s role. For instance, createPage() can only be executed if the user is an editor or an administrator, deletePage() only if the user is an administrator too.

Instead of spreading the logic of checking for a user’s role in both createPage and deletePage, we can use one function is Permitted() as below.

//get the current Subject

Subject current User = context.getSubject();

if (is Permitted(currentUser)) {

    //allow execution of deletePage

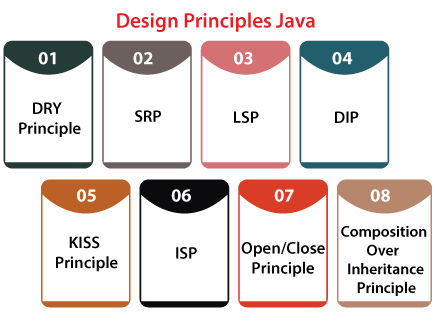
} else {

    //block execution

}

When you keep the logic of *is Permitted()* to one place, you avoid duplication and also enable re-use of the code. The added advantage is separation of logic i.e. *createPage()* and *deletePage()* don’t need to know how the permission is checked.

As always there is more than get to you every day.



Advantages of DRY

I- Maintainability

The biggest benefit of using DRY is maintainability.

If the logic of checking permission was repeated all over the code, it becomes difficult to fix issues that arise in the repeated code. When you fix a problem in one, you could easily forget to fix the problem in other occurrences.

Also, if you have to modify the logic, you have to copy-paste all over the place. By having a non-repeated code, you only have to maintain the code in a single place. New logic and bug fixes can be made one place instead of many. This leads to a robust and dependable software.

II- Readability

More often than not, DRY code is more readable. This is not because of the DRY principle itself, but rather because of the extra effort, the developer put into the code to make it follow certain principles such as DRY.

III- Reuse

DRY inherently promotes reuse of code because we are merging two or more instances of repeating code into a single block of code. The reusable code pays off in the long run as it speeds development time.

IV- Cost: If the management needs to be convinced regarding spending more time on improving the quality of code, this is it – More code can cost more. More code takes more people, more time to maintain and to address bugs. In addition, More time to develop and more bugs lead to a very unhappy customer.

V- Testing: We are not talking about manual testing; We are talking about unit tests and integration tests here. The more paths and functions you have to cover using the tests, the more code you have to write for tests. If code is not repeated, you just have to test one main path. Of course, different behaviors still need to be tested.