**Compile and Run Time Dependency**

****

**Here is the scenario we have:**

You can think of all the code below is created by you, meaning you are the owner and you already locked the files in Configuration Management System so that others cannot alter it without your information. And you are tired of randomly popping change requests for asking to use different sorting algorithms for the sort function.

**abstract** **class** Account {

// ...

// Many lines for generic bank account needs

// ...

// Default sorting method for transactions etc.

**public** **void** sort(){

System.***out***.println("Insertion Sort");

}

}

**class** Saving **extends** Account{

// ...

// Many lines for saving account needs

// ...

// Saving accounts likes to use Bubble sort for sorting needs.

@Override

**public** **void** sort(){

System.***out***.println("Bubble Sort");

}

}

**class** Debit **extends** Account{

// ...

// Many lines for debit account needs

// ...

//Inherits the base class sorting functions

}

**class** Dividend **extends** Account{

// ...

// Many lines for dividend account needs

// ...

@Override

**public** **void** sort(){

System.***out***.println("Bubble Sort");

}

}

// For demonstration purpose, we have only main method here

**public** **class** CompileRunTimeDependency{

**public** **static** **void** main(String[] args) {

Account accSaving = **new** Saving();

accSaving.sort(); //Prints: Bubble Sort

Account accDebit = **new** Debit();

accDebit.sort(); //Prints: Insertion Sort

Account accDividend = **new** Dividend();

accDividend.sort(); //Prints: Bubble Sort

/\* CONSOLE OUTPUT:

\* Bubble Sort

\* Insertion Sort

\* Bubble Sort

\*

\* \*/

}

}

**Current Picture:**

If a client creates a Saving account instance we know it will always use Bubble Sort, it is already set in stone. Or, a client creates Debit object it will always call Insertion sort. It is not possible to change sorting behavior without changing code, this is compile time dependency.

**Change Request Pops**

But a change request has just arrived wanting to use Insertion sort algorithm for Dividend. This is possible but we need to change the code - again compile time dependency. As said before you don't want anyone to change your existing code, including yourself!

**Let's Critique the Current Design**

We violate - "Single Responsibility Principle" Sorting is not the job of a bank account, bank account should do only bank account things.

Second: We have a code duplication, we have copy/paste Bubble sort algorithm in Saving and Dividend account.

Third: We cannot use different sorting algorithm at run time. We have to touch the code – compile time dependency.

**How to Accommodate the Change Request?**

We have compile-time dependency due to inheritance - therefore we have to modify the existing code to accommodate the change request above. And in the future, there would be a request for even additional sorting algorithms, such as Selection, Merge Sort etc, then life would be harder. Let’s make such a change which enables our client to use different sorting algorithms without asking us any code change.

**Refactoring Time**

First step: Take out each sorting algorithm to its own class, but wait a minute, we remember one design principle Program to Interface, this way we can break the compile-time dependency – we just added SortingServices interface, and we will reach run-time dependency, this will enable our clients to use different sorting algorithms as they want, without asking you to change your code.

**interface** SortingServices{

**void** sort();

}

**class** BubbleSort **implements** SortingServices{

**public** **void** sort(){

System.***out***.println("Bubble Sort");

}

}

**class** InsertionSort **implements** SortingServices{

**public** **void** sort(){

System.***out***.println("Insertion Sort");

}

}

We have just pulled out the sorting algorithms, added an interface to make our code has standard method names and take advantage of programming to interface to use the benefit of run-time dependency.

Now, we need to update Bank Account code to use the moved out sorting algorithms. For this, we create an instance of SortingServices interface and initialize it in sort() function with default function of InsertionSort.

**abstract** **class** Account {

//We added reference to interface

SortingServices sortService;

// ...

// Many lines for generic bank account needs

// ...

// Default sorting method for transactions etc.

**public** **void** sort(){

// Default sorting still Insertion

sortService = **new** InsertionSort();

sortService.sort();

//System.out.println("Insertion Sort");

}

}

We need to change in Saving and Dividend too, let's do it. In the constructor we initialized sortService to BubbleSort since Saving uses BubbleSort. We have just added setSortingAlgorithm to enable our client to change the sorting algorithm at runtime, this is the advantage of Program to Interface principle.

**class** Saving **extends** Account{

// ...

// Many lines for saving account needs

// ...

// Saving accounts likes to use Bubble sort for sorting needs.

**public** Saving(){

//Default sorting for Saving Account

sortService = **new** BubbleSort();

}

// We can change sorting algorithm at run-time

// We do not need to change our code anymore for sorting algorithm

**public** **void** setSortingAlgorithm(SortingServices sort){

**this**.sortService = sort;

}

@Override

**public** **void** sort(){

//Just call sort method

sortService.sort();

//System.out.println("Bubble Sort");

}

}

When we do the same update for Dividend, I noticed we have just made code duplication, setSortingAlgorithm method, we can pull-up that method to Account class to get rid of code duplication in Dividend and Saving.

**class** Dividend **extends** Account{

// ...

// Many lines for dividend account needs

// ...

**public** Dividend(){

//Default sorting for Saving Account

sortService = **new** BubbleSort();

}

@Override

**public** **void** sort(){

//Just call sort method

sortService.sort();

//System.out.println("Bubble Sort");

}

}

Updated Account class - we have just moved setSortingAlgorithm method to Account class, and removed from the derived classes (from Dividend and Saving classes)

**abstract** **class** Account {

//We added reference to interface

SortingServices sortService;

// ...

// Many lines for generic bank account needs

// ...

// We can change sorting algorithm at run-time

// We do not need to change our code anymore for sorting algorithm

**public** **void** setSortingAlgorithm(SortingServices sort){

**this**.sortService = sort;

}

// Default sorting method for transactions etc.

**public** **void** sort(){

// Default sorting still Insertion

sortService = **new** InsertionSort();

sortService.sort();

//System.out.println("Insertion Sort");

}

}

Tested and it works as before, we just finished our refactoring.

**Summary – What have we achieved?**

We have just used Strategy Pattern to break compile-time dependency and our code does not need to be changed anymore to swap a different sorting algorithm. Our code can be enhanced independently; a new sorting algorithm can be added without disturbing your code.  Our clients are happier; they do not need to beg you to change your code.  By programming to interface we break the compile time dependency to run time dependency.

In short, compile time dependency requires code to be changed, but run time dependency not.

Everyone is happy. Breaking dependency, reducing complexity, more flexible code.

Thanks for reading.