Sprint 4:

In this sprint, we updated our backend to do three optimizations and add some other extensions and new features based on the result of sprint3. For the three optimizations, now our compiler is able to delete dead (unused) code, combine loops with same conditions, and support human-readable list representation in Python target code. In order to do these optimizations, we slightly modified our data structure in the code generator. In terms of extensions to features, in addition to ArrayList we extended to support in Sprint3, now our compiler is able to support converting HashMap in Java to dictionary in Python to give user furthermore flexibility. Details are explained below.

Data Structure Modification:
The reason to modify our data structure that stores the whole target codes is that when we were implementing dead code elimination, we found that we can barely track these inserted codes and therefore we had no way to delete them. Thus, we decided to change storing all of our target codes in a 1D list to storing each class as a nested dictionary. The rule is that the outmost key is class and later put all these class dictionaries into a 1D list. Then inside each dictionary, we store each statement that has parentheses as a dictionary. The key is some specific names, like 'for_stmt1', 'while_cond', 'if_stmt1' so we can find the statement we want later easier. And the values are a dictionary containing inner statements.

With this modification, we can continue our optimizations.

Optimization1: Dead Code Elimination
This optimization deletes declared but unused variables inside each class. Our motivation to do this optimization is that being able to write clean codes is a good habit and is very important and thus this optimization can remind users, who are expected to be learners, know that their codes still have places to optimize. We support this optimization by keeping track of all variables declared in the class and counting the number of times they are called after declaration. When outputting the target code, if the specific variable is not ever called, it will be deleted and not output it.

Optimization2: Loop Fusion
This optimization merges while loops with same conditions and update rules in each class. We did this with the same motivation as our last optimization. This optimization is enabled by keeping track of all while loops condition in local variables. Then every new while loop will be matched to recorded loops by condition and update step. There is a helper function check_step to verify if the merge should continue or not.
We decided not to optimize for loops because the compiler does not support redeclaring a variable and therefore there is no way to have two for loops with the same condition.

Optimization3: ArrayList
In order to give users a better idea of what is inside the list, we decided to move forward to visualize it. So, our result will look like this [1, 2] which is much better than list.append(1), list.append(2). In terms of implementing it, instead of putting values directly into the target output list, we put it into a list inside a global dictionary which uses the list's name as the key and uses an empty list to occupy the position where the list should be. Therefore, we can keep track of all list values and not lose them. When output the result, we put the already appended list into the target code instead of the empty list.

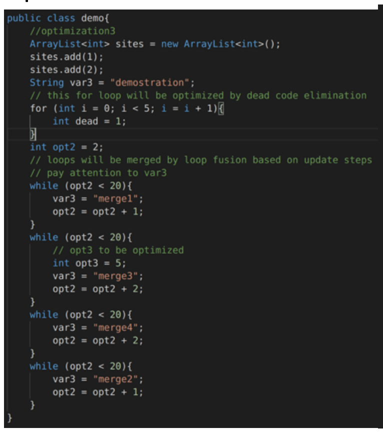
Extension - HashMap class data structure to Dictionary:

In terms of implementing it, we create new type checker and new declaration in symbol table for the map type in order to store variable types. We store the map type as a list, with HashMap itself, the key variable type, and the value variable type. We decide to convert the HashMap data type into Dictionary in the Python because of the similar data structure. also implement some methods: 1. Put(HashMap) -> directly assign value to the key(Dictionary) 2. Remove(HashMap) -> Del(Dictionary). 3. Clear(HashMap) -> Clear (Dictionary).

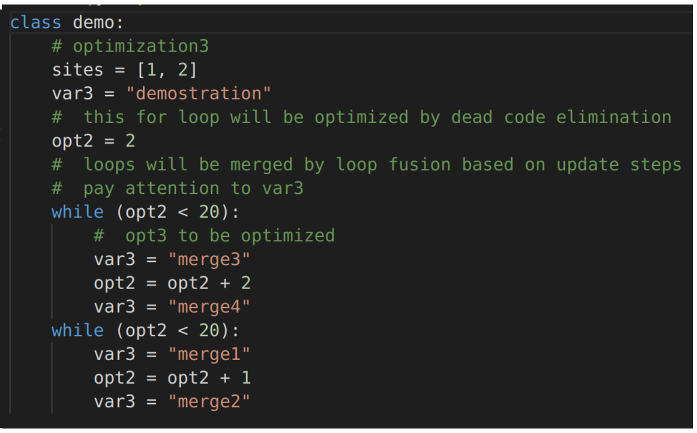
About running the script and verifying outputs:
Run bash file “sprint4.sh” in the terminal panel with the command “./sprint4.sh”, chmod to u+x if necessary.
The bash scripts will automatically execute several examples we provided and generate an executable python file with the example’s name.

Here are two examples. One is to mainly show how optimization3 works as optimization1 and 2 are demonstrated in the presentation. The other one is to show how the new feature HashMap works.

Optimization:

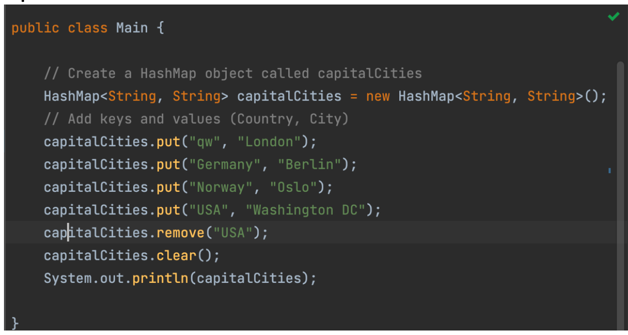


```java
public class demo{
    //optimization3
    ArrayList<int> sites = new ArrayList<int>();
    sites.add(1);
    sites.add(2);
    String var3 = "demostration";
    // this for loop will be optimized by dead code elimination
    for (int i = 0; i < 5; i = i + 1){
        int dead = 1;
    }
    int opt2 = 2;
    // loops will be merged by loop fusion based on update steps
    // pay attention to var3
    while (opt2 < 20){
        var3 = "merge1";
        opt2 = opt2 + 1;
    }
    while (opt2 < 20){
        // opt3 to be optimized
        int opt3 = 5;
        var3 = "merge3";
        opt2 = opt2 + 2;
    }
    while (opt2 < 20){
        var3 = "merge4";
        opt2 = opt2 + 2;
    }
    while (opt2 < 20){
        var3 = "merge2";
        opt2 = opt2 + 1;
    }
}
```



```python
class demo:
    # optimization3
    sites = [1, 2]
    var3 = "demostration"
    #  this for loop will be optimized by dead code elimination
    opt2 = 2
    #  loops will be merged by loop fusion based on update steps
    #  pay attention to var3
    while (opt2 < 20):
        #  opt3 to be optimized
        var3 = "merge3"
        opt2 = opt2 + 2
        var3 = "merge4"
    while (opt2 < 20):
        var3 = "merge1"
        opt2 = opt2 + 1
        var3 = "merge2"
```

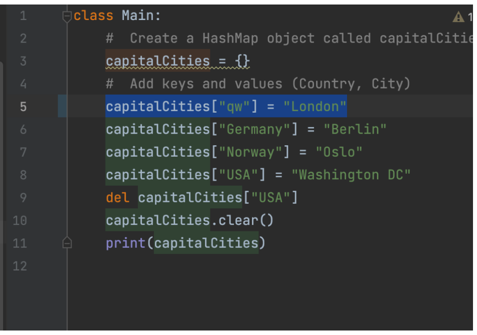
In the left is the Java code. We can clearly see there is a arraylist declaration and its add operation in the top and there is a for loop to be eliminated because of dead code as well as four while loops to be merged by loop fusion based on their update steps.
The code in the right is the output. It is obviously shorter and cleaner and it has completed all optimizations It should do.



```java
public class Main {

    // Create a HashMap object called capitalCities
    HashMap<String, String> capitalCities = new HashMap<String, String>();
    // Add keys and values (Country, City)
    capitalCities.put("qw", "London");
    capitalCities.put("Germany", "Berlin");
    capitalCities.put("Norway", "Oslo");
    capitalCities.put("USA", "Washington DC");
    capitalCities.remove("USA");
    capitalCities.clear();
    System.out.println(capitalCities);

}
```



```python
class Main:
    #  Create a HashMap object called capitalCitie
    capitalCities = {}
    #  Add keys and values (Country, City)
    capitalCities["qw"] = "London"
    capitalCities["Germany"] = "Berlin"
    capitalCities["Norway"] = "Oslo"
    capitalCities["USA"] = "Washington DC"
    del capitalCities["USA"]
    capitalCities.clear()
    print(capitalCities)
```

In the left is the input java code with HashMap example, and the right one is target python Dictionary output code.