Program Analysis, University of Stuttgart, Winter 2023/2024

Exercise 4: Information Flow and Call Graphs
—**Solution**—

Deadline for uploading solutions via Ilias:
January 18, 2024, 11:59pm Stuttgart time

Task 1 Information Flow Analysis [32 points]

This task is about dynamic information flow analysis. Consider the following JavaScript code to analyze:

```
1   let paymentMethod = getPaymentMethod();
2   let paymentPassword = getPassword(paymentMethod)
3   let paymentInfo = shopping(paymentMethod, paymentPassword);
4
5   if (paymentInfo[3] == false) {
6     console.alert("Error.");
7     // use another paymentMethod
8     newPaymentMethod = changePaymentMethod(paymentMethod);
9     paymentMethod = newPaymentMethod;
10    paymentPassword = getPassword(paymentMethod)
11    paymentInfo = shopping(paymentMethod, paymentPassword);
12  }
13
14  let userId = getUserId();
15  if (paymentMethod == "card") {
16    if (paymentInfo[2] > 100) {
17      sendToThirdParty(userId, paymentInfo[0]);
18    }
19  } else {
20    console.log("Purchased: ${paymentInfo[1]}")
21  }
```

This code snippet illustrates the sequential steps a user follows to make a purchase on an online mall. It assumes two payment methods: card and Paypal. In the event of an initial payment failure, an alternative payment method will be attempted to facilitate the continuation of the transaction. Function `changePaymentMethod` is used to switch the payment methods. Table 1 lists the details of payment methods (password, and whether it gets a sufficient balance) and the information of purchased products of two users. Assume both of them start payment with card.

Table 1: Details on payment methods and products

userId	Card	Paypal	Product	Price
1	passwordA, sufficient	passwordB, sufficient	Product1	139
2	passwordC, insufficient	passwordD, sufficient	Product2	20

The functions `console.alert`, `sendToThirdParty` and `console.log` are untrusted sinks that should be reached by public information only. The function `shopping` returns an array of four values: `[cardPaypalNumber, product, price, isPaid]`. The elements of an array can have different security labels. There are four security classes for this program, which are presented in the lattice in Figure 1.

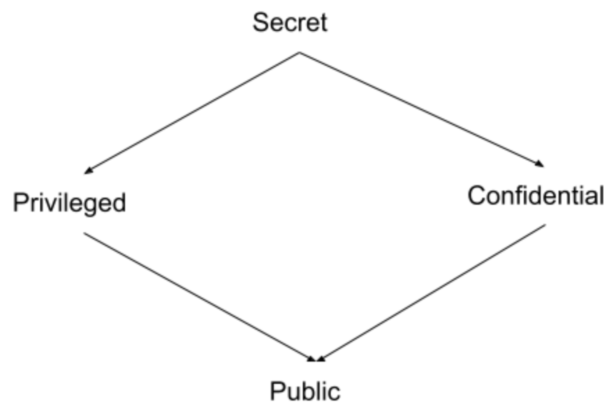


Figure 1: Lattice of security labels for Task 1.

The variable and the values returned by the functions are labeled as follows:

- paymentPassword: Secret.
- shopping: Returns an array of four elements:

  Element 1. cardPaypalNumber: Secret

  Element 2. product: Public

  Element 3. price: Privileged

  Element 4. isPaid: Privileged
- getPaymentMethod: Returns a Public value.
- changePaymentMethod: Returns a Public value.
- getUserId: Returns a Secret value.

Subtask 1.1 User 1 [16 points]

Consider a dynamic information flow analysis that considers both explicit and implicit flows. The information policy is that only Public information should reach the untrusted sinks.

- What are the security labels of variables and expressions (after executing the given line) during the execution? Use the following template to provide your answer. For unreachable lines of code during this execution, fill the security label with *Unreachable*.

  Solution:

Line	Variable or expression	Security label
1	`paymentMethod`	*Public*
2	`paymentPassword`	*Secret*
3	`paymentInfo`	*paymentInfo[0]: Secret,* *paymentInfo[1]: Public,* *paymentInfo[2]: Privileged,* *paymentInfo[3]: Privileged*
5	`paymentInfo[3] == false`	*Privileged*
8	`newPaymentMethod`	*Unreachable*
14	`userId`	*Secret*
16	`paymentInfo[2] > 100`	*Privileged*

- Does the execution violate the information flow policy? Explain your answer.

  The execution reaches an untrusted sink (sendToThirdParty) after executing line 17. It gets the values `userId` and `paymentInfo[0]`, which are Secret. Conclusion: The execution violates the policy.

- If the answer to the previous question was yes: How can you modify the line(s) of code causing the violation of the policy so that you reduce information leakage?

  Avoid using the untrusted sink (sendToThirdParty).

- Assume an attacker does not know the source code of the program but sees all values passed as arguments to the untrusted sinks. Does the attacker learn anything about the `cardPaypalNumber` of User 1?

  Even without the knowing the source code, the attacker learns the `cardPaypalNumber`, because it is passed to `sendToThirdParty`.

Subtask 1.2 User 2 [16 points]

Consider a dynamic information flow analysis that considers both explicit and implicit flows. Again, the information policy is that only Public information should reach the untrusted sinks.

- What are the security labels of variables and expressions (after executing the given line) during the execution? Use the following template to provide your answer. For unreachable lines of code during this execution, fill the security label with *Unreachable*.

  Solution:

Line	Variable or expression	Security label
1	`paymentMethod`	*Public*
2	`paymentPassword`	*Secret*
3	`paymentInfo`	*paymentInfo[0]: Secret,* *paymentInfo[1]: Public,* *paymentInfo[2]: Privileged,* *paymentInfo[3]: Privileged*
5	`paymentInfo[3] == false`	*Privileged*
8	`newPaymentMethod`	*Privileged*
14	`userId`	*Secret*
16	`paymentInfo[2] > 100`	*Unreachable*

- Does the execution violate the information flow policy? Explain your answer.

  The execution reaches to two untrusted sinks (`console.alert` in line 6, `console.log` in line 20). The branch that line 6 depends on Privileged data (`paymentInfo[3] == false`, which has label Privileged $\bigoplus$ Public = Privileged). This causes an implicit flow to `console.alert`, which means the execution violates the policy. The `console.log` in line 20 is reached by Public information only, however, it also involves in an implicit flow. When the user pays with Paypal, the execution of line 20 will leak the privileged data in lines 5 and 15.

- If the answer to the previous question was yes, how can you modify the line(s) of code causing the policy violtion so that you reduce information leakage?

  Avoid using the untrusted sinks (`console.alert, console.log`) at lines 6 and 20.

- Assume an attacker who does not know the source code but sees all values passed as arguments to the untrusted sinks. Does the attacker learn anything about whether User 2 has paid for the product with the payment method the user has selected first?

  No, because the message passed to `console.alert` is a generic message it does not reveal anything about the user's ability to pay the requested amount.

- Now assume an attacker who knows the source code and also sees all values passed as arguments to the untrusted sinks. Does the attacker learn anything about whether User 2 has paid for the product with the payment method the user has selected first?

Yes, the attacker learns that the user was not able to pay the amount, because the message `"Error.` is passed to `console.alert` if and only if `paymentInfo[3] == false` evaluates to `false`.

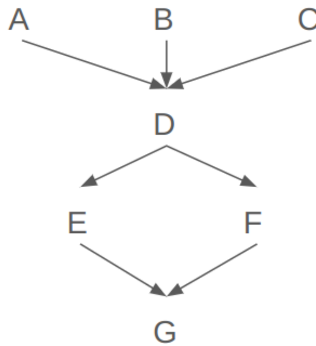
Task 2 Universally Bounded Lattice [12 points]

In this task, we will analyze the characteristics and properties of universally bounded lattices.

Subtask 2.1 Draw the defined lattices [4 points]

Consider a policy defined with the following ordering rules: `A > D`, `B > D`, `C > D`, `D > E`, `D > F`, `E > G`, `F > G`, where A, B, C, D, E, F and G are corresponding security labels.

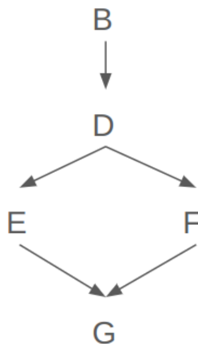
- Draw the graph of the previously defined Lattice.



- Is it a universally bounded lattice (Explain)?

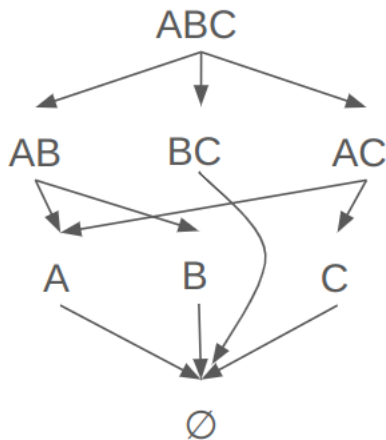
  No. It is impossible to define a least upper bound operator.

- Consider a policy that only uses the labels B, D, E, F, G (with same previous ordering rules). Is it a universally bounded lattice (Explain)? Yes, because it has all the necessary characteristics: – A limited set of security classes – A partial order – A lower bound – An upper bound – A least upper bound operator – A greatest lower bound operator



Subtask 2.2 Characteristics [8 points]

Consider the following structure that represents a universally bounded lattice:



Answer the following questions:

- Give the set S of security classes.

  Solution:
  S = ABC, AB, BC, AC, A, B, C, $\emptyset$

- What is the lower bound $\bot$?

  Solution:
  $\bot = \emptyset$

- What is the upper bound $\top$?

  Solution:
  $\top$ = ABC

- Let $\bigoplus$ be the least upper bound operator. What is the result of the following operations?

  Solution:
  $B \bigoplus AB$ = AB

  $ABC \bigoplus \emptyset$ = ABC

  $B \bigoplus C$ = ABC

- Let $\bigotimes$ be the greatest lower bound operator. What is the result of the following operations?

  Solution:
  $C \bigotimes AC$ = C

  $ABC \bigotimes A$ = A

Task 3 Call Graphs: CHA, RTA and VTA [56 points]

Consider the following class diagram of a Java program:

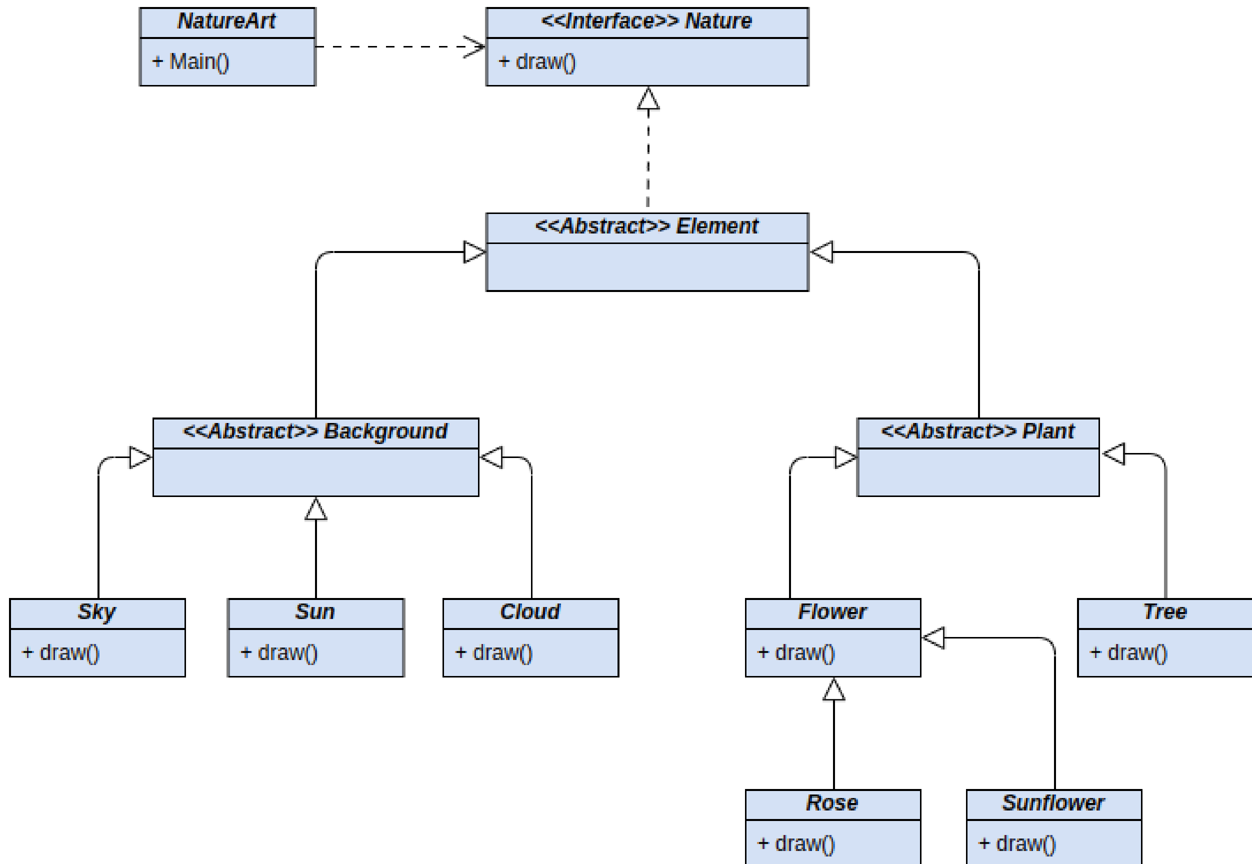


Figure 2: Class Diagram

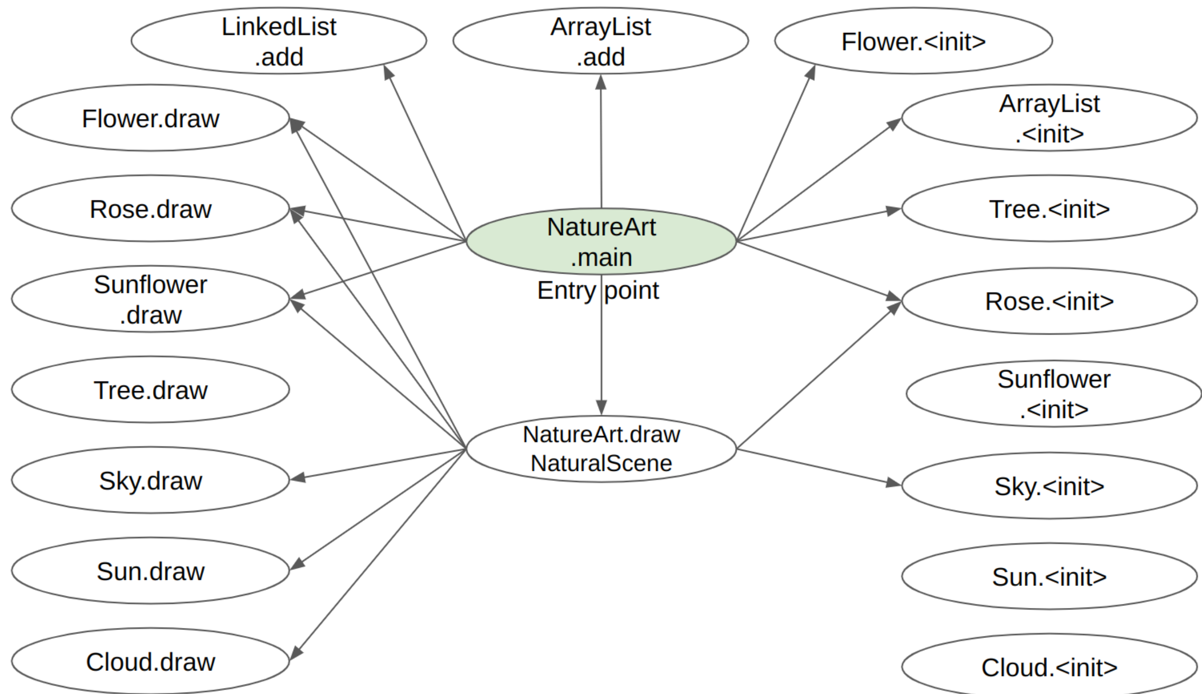
The implementation of the class `NatureArt` is presented in the snippet of code below. All the classes and interfaces presented in the diagram are in a package called model. Thus, line 3 (in the code) imports all of them.

```
1  import java.util.ArrayList;
2  import java.util.List;
3  import model.*
4
5  class NatureArt {
6    public static void main(String[] args) {
7      Flower flower = new Flower();
8      Tree oakTree = new Tree();
9      Rose rose1 = new Rose();
10     flower = (Flower)rose1;
11
12     List<Plant> plants = new ArrayList<Plant>();
13     plants.add(flower);
14     plants.add(oakTree);
15
16     flower.draw();
17     drawNaturalScene();
18   }
19
```

```
20    public static void drawNaturalScene() {
21      Background sky = new Sky();
22      Flower rose2 = new Rose();
23
24      sky.draw();
25      rose2.draw();
26    }
27  }
```

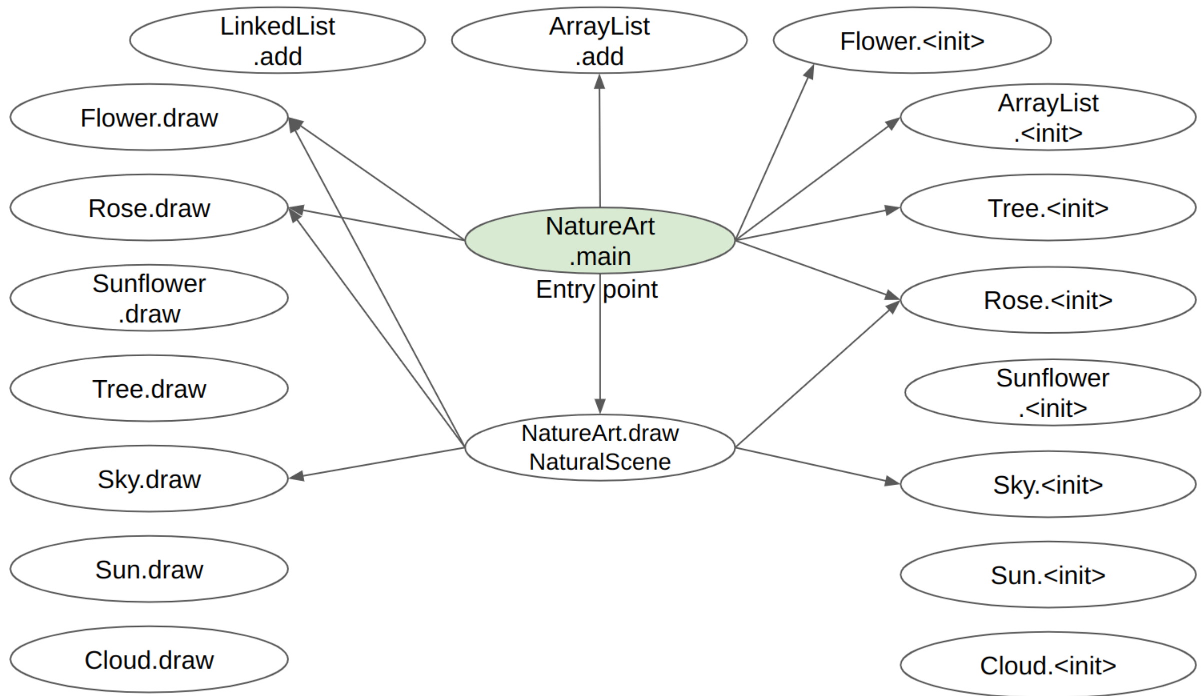
Subtask 3.1 CHA Graph [8 points]

- Considering the previous class diagram in Figure 2 and the snippet of code, provide the call graph computed by the CHA (Class Hierarchy Analysis) algorithm. In this and the following tasks, you can ignore any call graph nodes that are not given in the template.



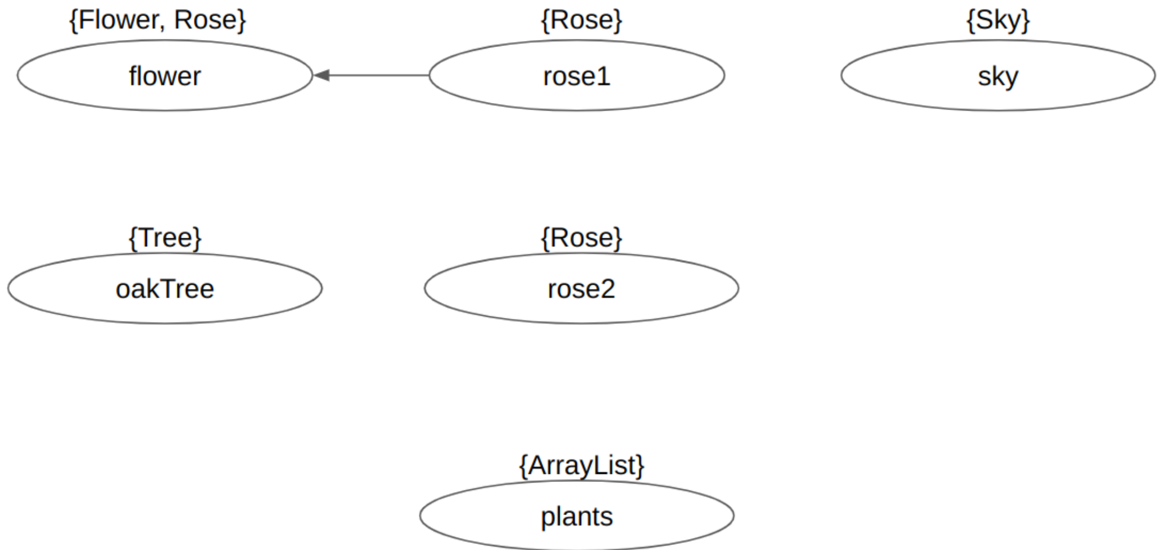
Subtask 3.2 RTA Graph [8 points]

- Considering the previous class diagram in Figure 2 and the snippet of code, provide the call graph computed by the RTA (Rapid Type Analysis) algorithm.

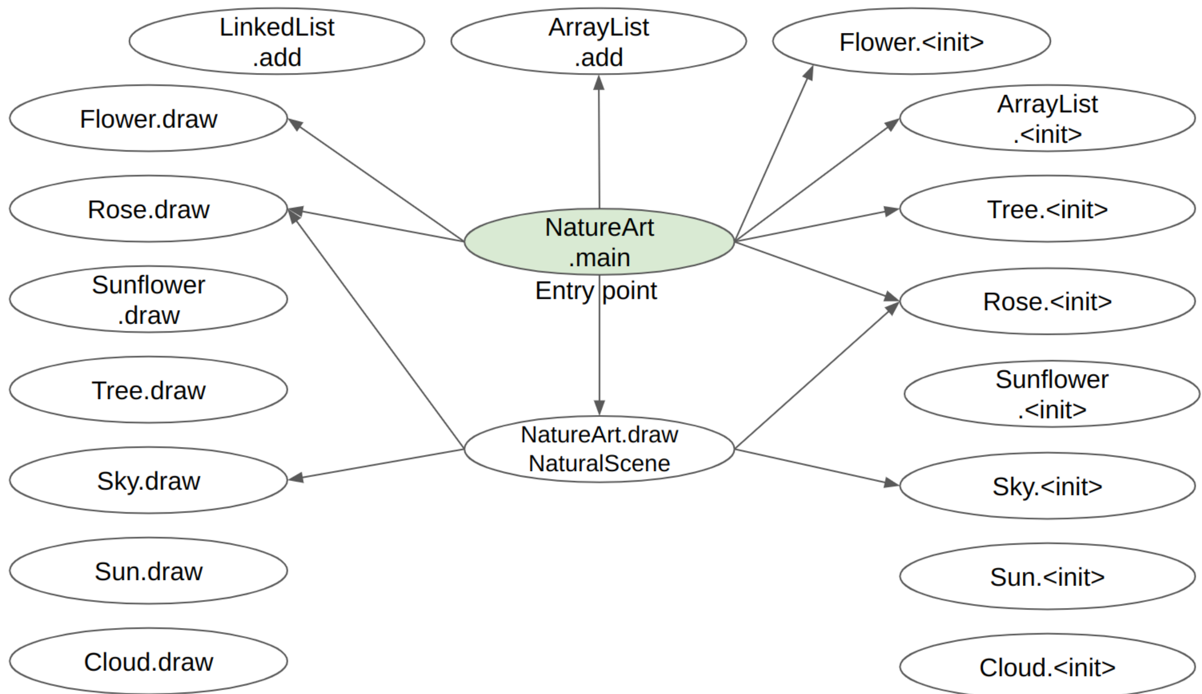


Subtask 3.3 VTA Graph [14 points]

- Considering the previous class diagram in Figure 2 and the snippet of code, provide the type propagation graph computed by VTA (Variable Type Analysis).

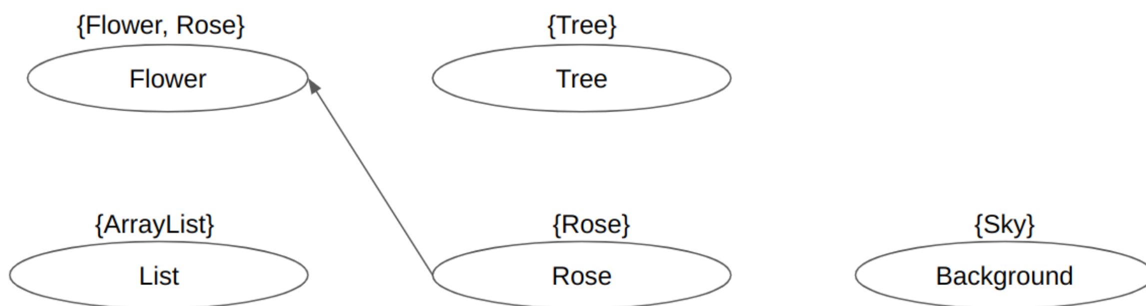


- Based on the types computed by VTA, give the call graph that VTA produces starting from the RTA graph.

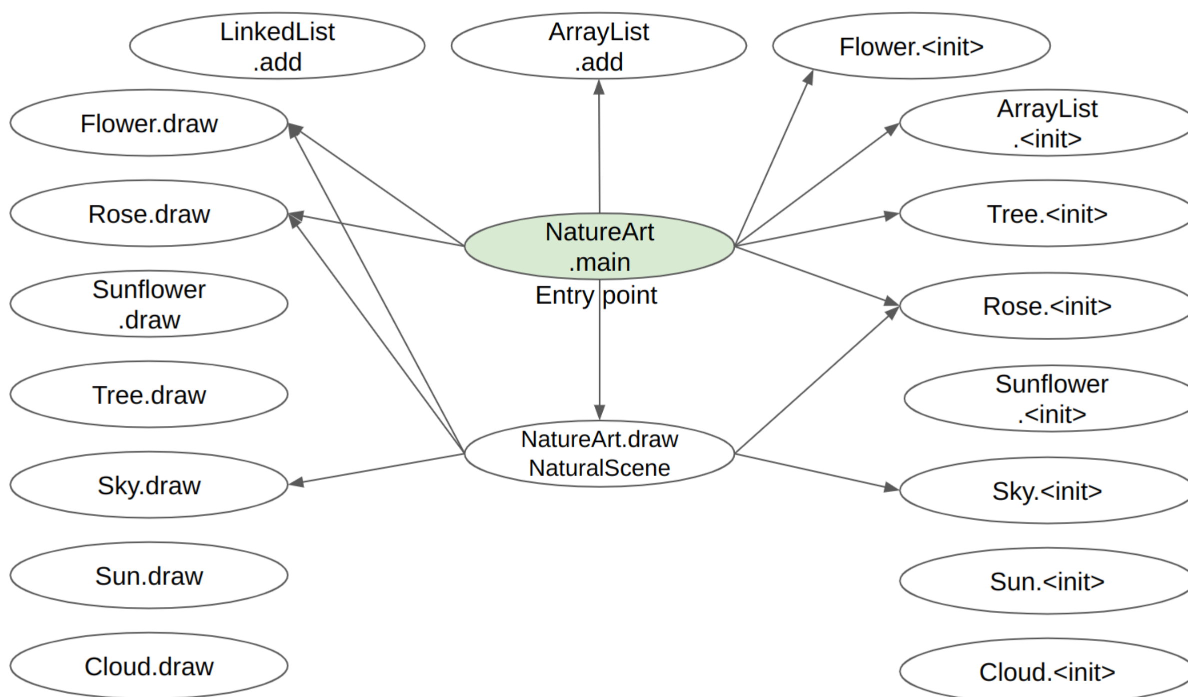


Subtask 3.4 DTA Graph [14 points]

- Considering the previous class diagram in Figure 2 and the snippet of code, provide the type propagation graph computed by DTA (Declared Type Analysis).

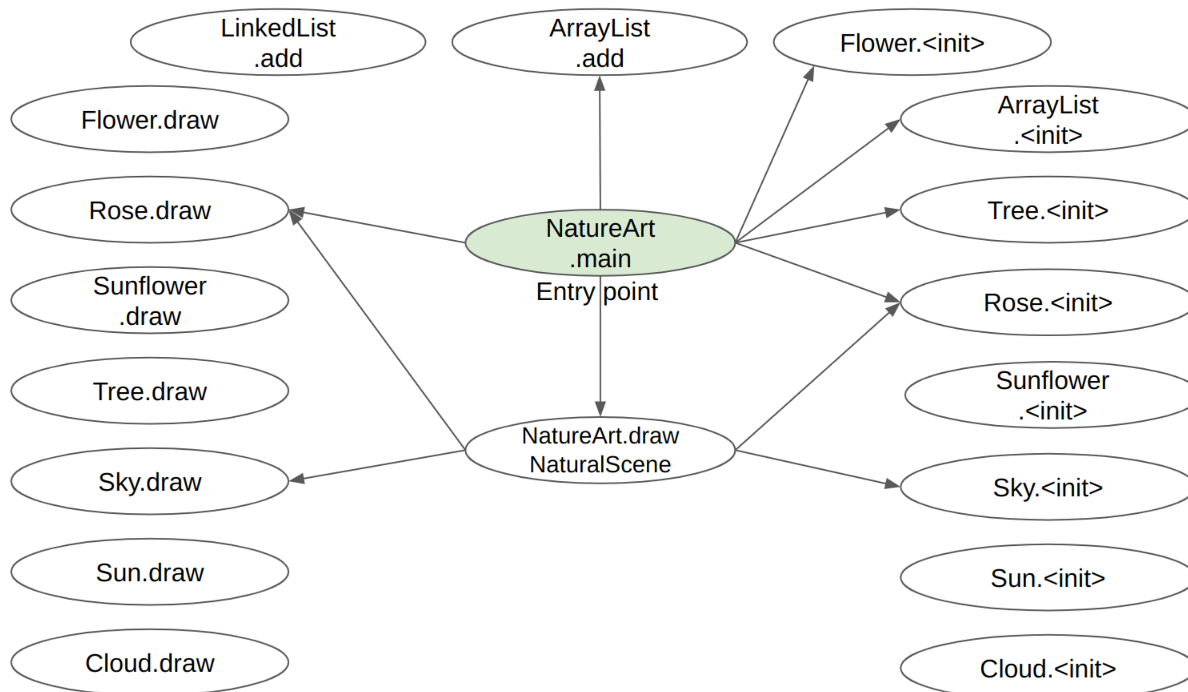


- Based on the types computed by DTA, provide the call graph.



Subtask 3.5 Dynamic Execution Call Graph [8 points]

- By performing a dynamic execution of the previous program, provide the call graph representing only the calls that happen during the dynamic execution.



Subtask 3.6 Comparision Between Algorithms [4 points]

- Using previously computed graphs, fill in the following table (Useless edges are edges that don't appear in the graph computed from dynamic execution):

Algorithm	Total number of edges	Number of useless edges
CHA	18	7
RTA	13	2
VTA	12	1
DTA	13	2