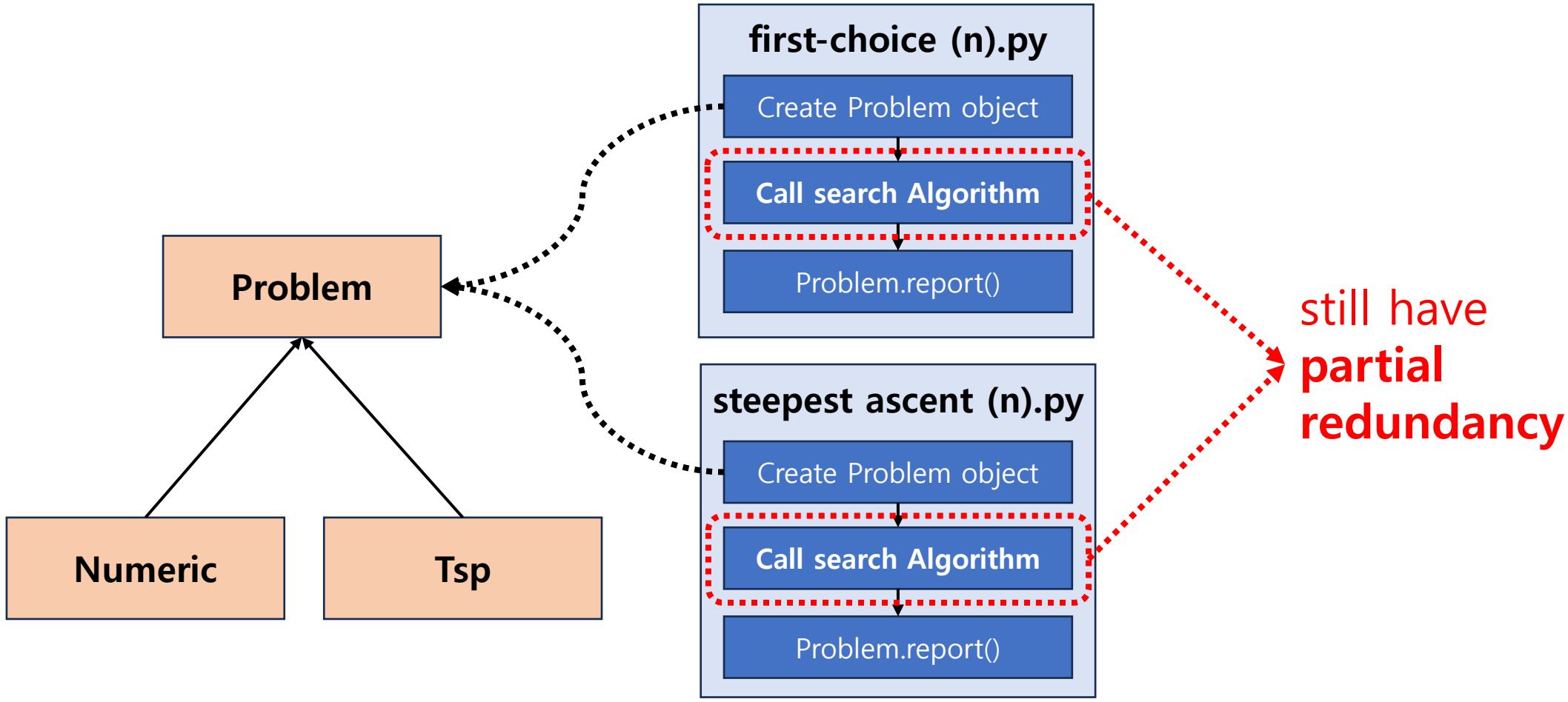
AI Programming
[Week 11] Practice

2024. 11. 14.

목차

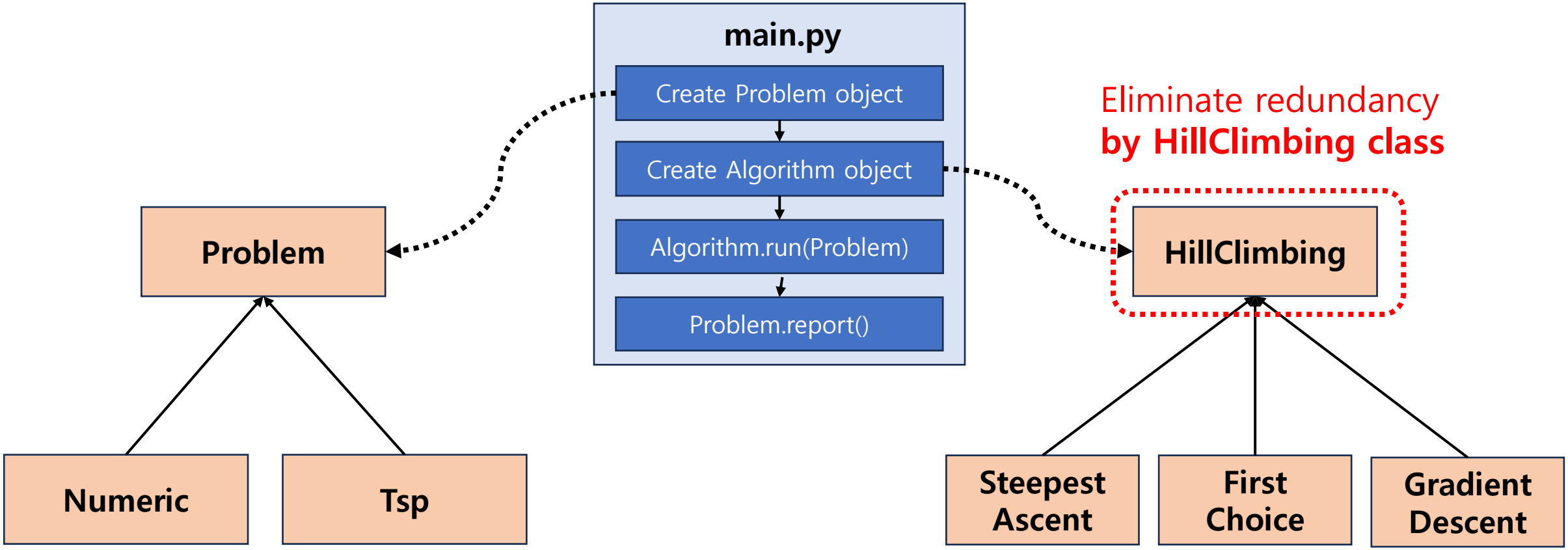
HW06 본인 제출 파일 준비

- first-choice (n).py
- first-choice (tsp).py
- steepest ascent (n).py
- steepest ascent (tsp).py
- problem.py
- gradient descent.py

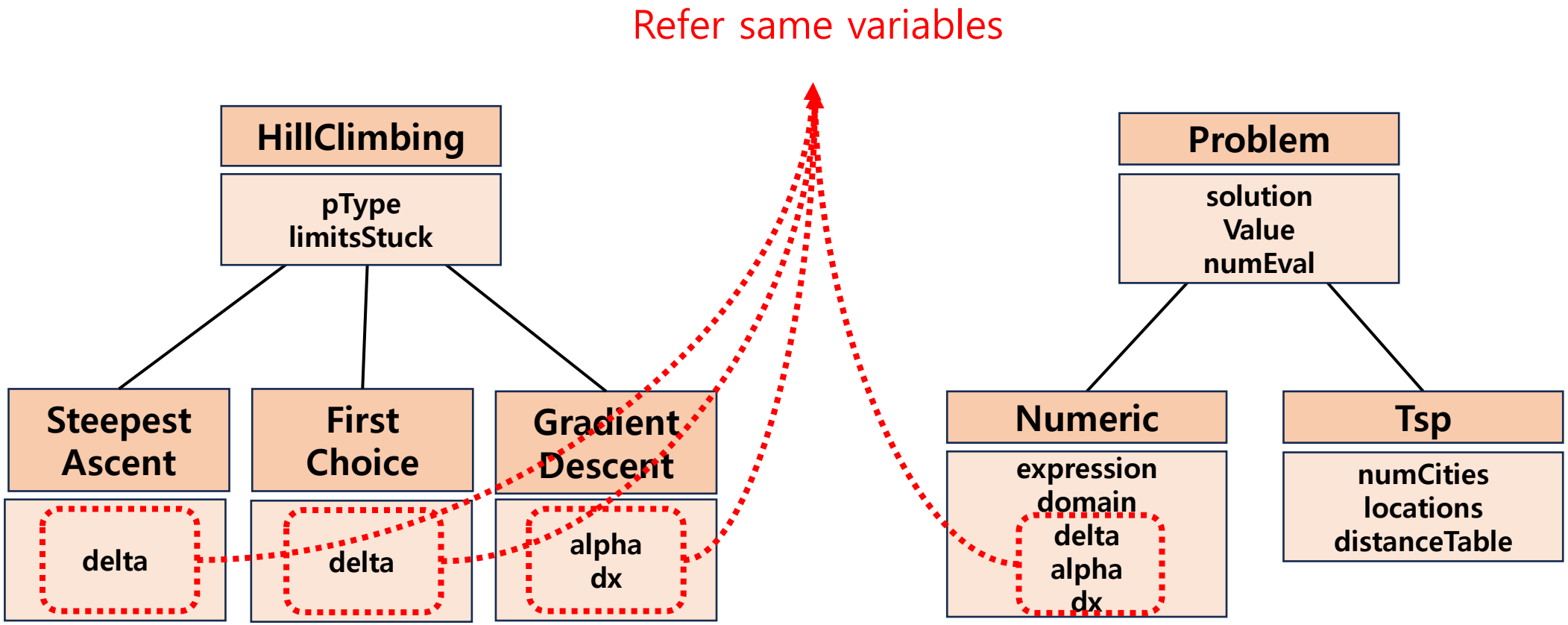


Previous program from the HW06 still has some redundancy. This can make program less readable or harder to manage.

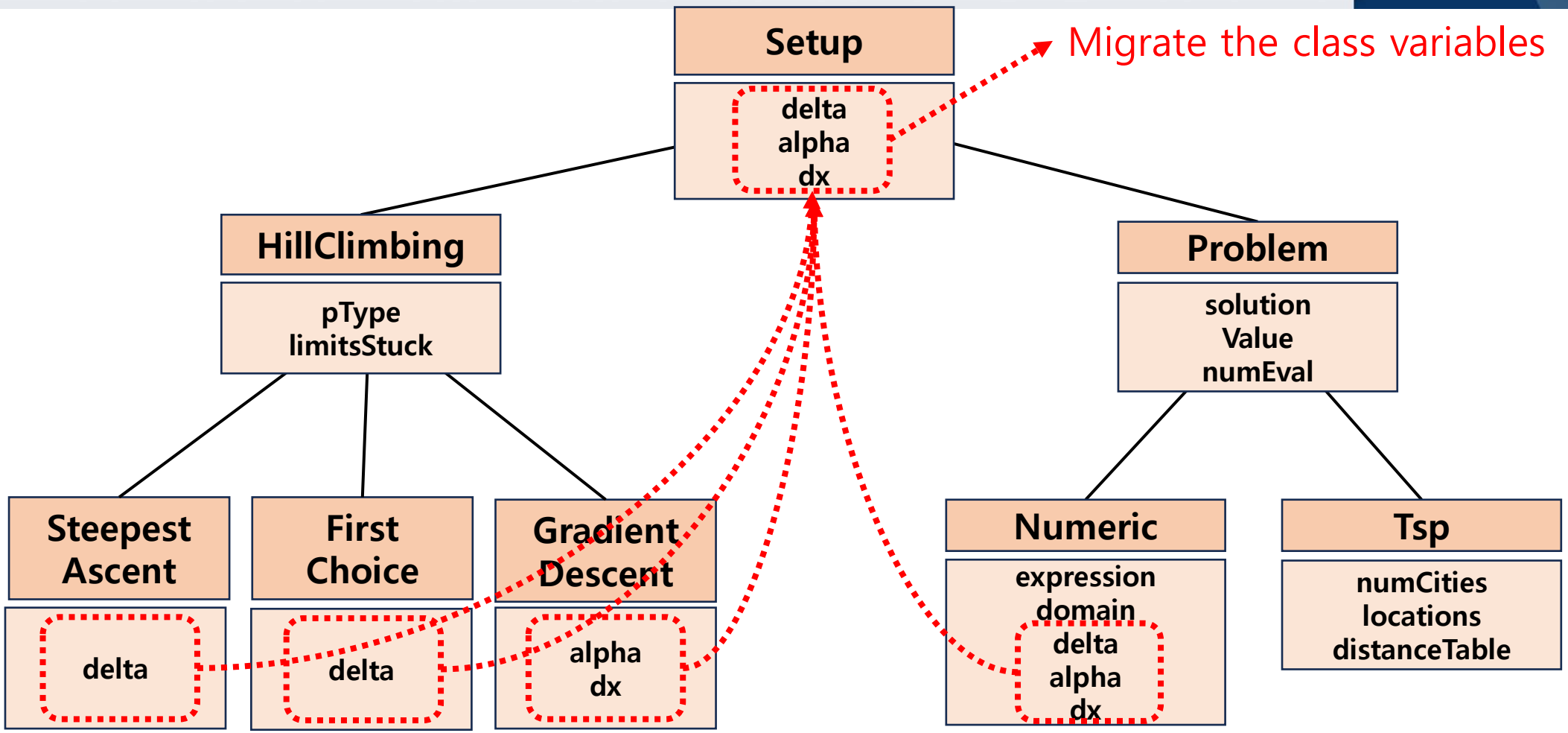
실습 목표



Eliminate redundancy of the code by defining **algorithm classes** that inherit from **HillClimbing class** and unite all the programs into **single main program**



Since **delta, alpha** and **dx** are needed by both the classes **HillClimbing** and **Problem**. We define a new class named **Setup** to hold those variables and make it a parent class of both



Since **delta, alpha** and **dx** are needed by both the classes **HillClimbing** and **Problem**. We define a new class named **Setup** to hold those variables and make it a parent class of both

1) new Superclass 'Setup' (setup.py)

- new Superclass 'Setup' : parent of 'Problem' and 'HillClimbing'

2) new class 'HillClimbing' (optimizer.py)

- search algorithms become subclasses under the 'HillClimbing'
- move 'displaySetting' to the 'HillClimbing' class and distribute 'run' method of the corresponding subclass

3) single program (main.py)

- new user interface to query problem type and algorithm type

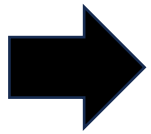
setup.py 정의, problem.py - Problem/Numeric Class 수정 (5분)

setup.py

```
class Setup:
    def __init__(self):
        self._delta = 0.01      # Step size for axis-parallel mutation
        self._alpha = 0.01      # Update rate for gradient descent
        self._dx = 10 ** (-4)   # Increment for calculating derivative
```

problem.py

```
class Problem:
    def __init__(self):
        self._solution = []
        self._value = 0
        self._numEval = 0
```



```
from [ ] import [ ]

class Problem([ ]):
    def __init__(self):
        [ ].__init__(self)
        self._solution = []
        self._value = 0
        self._numEval = 0
```

*Numeric class에서 delta alpha, dx 삭제해주기

Optimizer.py – HillClimbing 정의 (5분)

```
class HillClimbing(Setup):

    def __init__(self):
        # 1. Setup에 정의된 delta, alpha, dx에 접근하기 위해 Setup 초기화
        # 2. self._pType 정의하기 (Tsp인지, Numeric인지 구분하기 위한 Integer 변수 선언)
        # 3. self._limitStuck 정의하기 (지금은 First-choice에서만 사용하지만, 앞으로 추가될
        #    다른 hillclimbing 알고리즘에서 사용함)

    def setVariables(self, pType):
        # 1. pType을 인자로 받아서 self._pType에 assign

    def displaySetting(self):
        # 1. pType==1 (Numeric) 일 때만, Mutation step size를 출력하는 함수
        # first-choice.py 코드의 'print( " Mutation step size: " , p.getDelta())' 부분 활용

    def run(self):
        pass
```

Optimizer.py – FirstChoice 정의 (5분)

```
class FirstChoice(HillClimbing):
    def displaySetting(self):
        # first-choice.py 코드의 displaySetting 부분 활용
        # HillClimb에 정의했던 displaySetting을 Super를 통해 호출해서 구현하기

    def run(self, p):
        # first-choice.py에 정의했던 firstchoice 함수를 활용해서 구현
        # global Variable 대신 class variable을 활용하도록 변경
```

main.py 실습


```
Select the problem type:
  1. Numerical Optimization
  2. TSP
Enter the number: 1
Enter the file name of a function: problem/Ackley.txt

Select the search algorithm:
  1. Steepest-Ascent
  2. First-Choice
  3. Gradient Descent
Enter the number: 2

Objective function:
20 + math.e - 20 * math.exp(-(1/5) * ...

Search space:
 x1: (-30.0, 30.0)
 ...
 x5: (-30.0, 30.0)

Search Algorithm: First-Choice Hill Climbing

Mutation step size: 0.01
Max evaluations with no improvement: 100 iterations

Solution found:
(-11.994, -11.999, -8.999, 16.993, 0.996)
Minimum value: 17.986

Total number of evaluations: 532
```

Main 함수의 역할

1. 어떤 유형(TSP, Numeric)의 어떤 문제(tsp30.txt, Ackley.txt, ...)를 풀 것인지 입력 받기
2. 어떤 알고리즘으로 풀지 입력 받기
3. Problem/Optimizer Class를 이용해서 문제풀기
4. 결과 출력하기

main – selectProblem 정의 (5분)

```
from problem import *
from optimizer import *

def main():
    p, pType = selectProblem()

def selectProblem():
    print("Select the problem type:")
    print("  1. Numerical Optimization")
    print("  2. TSP")

    # 1 (Numeric) 또는 2 (TSP)를 입력 받아서 대응되는 Problem Class를 초기화해서 반환하기

    return p, pType
```

main – selectAlgorithm 정의 (8분)

```
from problem import *
from optimizer import *

def main():
    p, pType = selectProblem()
    alg = selectAlgorithm(pType) # 추가

def selectAlgorithm(pType):
    print()
    print("Select the search algorithm:")
    print("  1. Steepest-Ascent")
    print("  2. First-Choice")
    print("  3. Gradient Descent")

    # pType == 2 (TSP)일 경우, Gradient Descent를 입력 받으면 사용자로부터 재입력 받도록 구현
    # pType과 aType이 올바르게 설정 됐는지 확인하기 위한 invalid(pType, aType) 함수 추가 구현

    return alg
```

main – 알고리즘 수행 및 결과 출력 (5분)

```
from problem import *
from optimizer import *

def main():
    p, pType = selectProblem()
    alg = selectAlgorithm(pType)

    # Call the search algorithm
    alg.[   ](p)
    # Show the problem solved
    p.[        ]()
    # Show the algorithm settings
    alg.[              ]()
    # Report results
    p.[      ]()
```

algorithm instance와 problem instance의 method 들을 적절히 호출하여 알고리즘을 수행하고 결과를 출력할 수 있도록 구현

main 실행해보기 (3분 ~)

main.py 실행해서 Numeric 문제를 First-choice 알고리즘으로 풀어보기

과제 안내

Gradient Descent, Steepest Ascent를 OOP style로 구현하기

제출물:

파이썬 파일 총 4개를 **HW07_NAME 폴더**로 묶어서 **압축**하여 제출 (.zip)

- main.py
- optimizer.py
- problem.py
- setup.py

리포트 제출 (**.pdf**)

- 본론에는 각 알고리즘을 문제 유형(tsp, numeric)마다 실행시킨 결과 총 5개의 terminal screen shot 포함
- ***tsp-gradient descent**를 선택했을 때 나는 **예외처리 예시 추가** 포함
- (문제는 자유롭게 선택하세요.)