# Текст программы

## main.py

from src.biquadratic\_solver import BiquadraticSolver  
from src.input\_handler import InputHandler  
  
  
class BiquadraticEquationApp:  
 def run(self) -> None:  
 input\_handler = InputHandler()  
 solver = BiquadraticSolver()  
  
 try:  
 a, b, c = input\_handler.get\_coefficients()  
 result = solver.solve(a, b, c)  
 if len(result) == 0:  
 print("There are no real valued solutions.")  
 else:  
 print(f"The solutions are: {result}")  
 except Exception as e:  
 print(f"An error occurred: {e}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app = BiquadraticEquationApp()  
 app.run()  
from src.biquadratic\_solver import BiquadraticSolver  
from src.input\_handler import InputHandler  
  
  
class BiquadraticEquationApp:  
 def run(self) -> None:  
 input\_handler = InputHandler()  
 solver = BiquadraticSolver()  
  
 try:  
 a, b, c = input\_handler.get\_coefficients()  
 result = solver.solve(a, b, c)  
 if len(result) == 0:  
 print("There are no real valued solutions.")  
 else:  
 print(f"The solutions are: {result}")  
 except Exception as e:  
 print(f"An error occurred: {e}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app = BiquadraticEquationApp()  
 app.run()

## biquadratic\_solver.py

from math import sqrt  
from typing import Set  
  
  
class BiquadraticSolver:  
 def solve(self, a: float, b: float, c: float) -> Set[float]:  
 D = b\*\*2 - 4 \* a \* c  
  
 roots = set()  
  
 if a == 0:  
 raise Exception("Equation must be biquadratic (a != 0).")  
  
 if D >= 0:  
 quadratic\_roots = [  
 (-b - sqrt(D)) / (2 \* a),  
 (-b + sqrt(D)) / (2 \* a),  
 ]  
  
 for root in quadratic\_roots:  
 if root >= 0:  
 roots.add(sqrt(root))  
 roots.add(-sqrt(root))  
 return roots  
from math import sqrt  
from typing import Set  
  
  
class BiquadraticSolver:  
 def solve(self, a: float, b: float, c: float) -> Set[float]:  
 D = b\*\*2 - 4 \* a \* c  
  
 roots = set()  
  
 if a == 0:  
 raise Exception("Equation must be biquadratic (a != 0).")  
  
 if D >= 0:  
 quadratic\_roots = [  
 (-b - sqrt(D)) / (2 \* a),  
 (-b + sqrt(D)) / (2 \* a),  
 ]  
  
 for root in quadratic\_roots:  
 if root >= 0:  
 roots.add(sqrt(root))  
 roots.add(-sqrt(root))  
 return roots

## input\_handler.py

import sys  
from typing import Tuple  
  
  
class InputHandler:  
 def get\_float\_from\_terminal(self) -> float:  
 valid\_input = False  
 while not valid\_input:  
 inp = input()  
 try:  
 result = float(inp)  
 valid\_input = True  
 except Exception as e:  
 print(f"Invalid input. {e}. Please try again: ")  
 return result  
  
 def get\_coefficients(self) -> Tuple[float, float, float]:  
 if len(sys.argv) < 3:  
 print("Input coefficients:")  
  
 a = self.get\_float\_from\_terminal()  
 b = self.get\_float\_from\_terminal()  
 c = self.get\_float\_from\_terminal()  
 else:  
 try:  
 args = sys.argv[1:]  
 a = float(args[0])  
 b = float(args[1])  
 c = float(args[2])  
 except Exception as e:  
 raise e  
 return a, b, c  
import sys  
from typing import Tuple  
  
  
class InputHandler:  
 def get\_float\_from\_terminal(self) -> float:  
 valid\_input = False  
 while not valid\_input:  
 inp = input()  
 try:  
 result = float(inp)  
 valid\_input = True  
 except Exception as e:  
 print(f"Invalid input. {e}. Please try again: ")  
 return result  
  
 def get\_coefficients(self) -> Tuple[float, float, float]:  
 if len(sys.argv) < 3:  
 print("Input coefficients:")  
  
 a = self.get\_float\_from\_terminal()  
 b = self.get\_float\_from\_terminal()  
 c = self.get\_float\_from\_terminal()  
 else:  
 try:  
 args = sys.argv[1:]  
 a = float(args[0])  
 b = float(args[1])  
 c = float(args[2])  
 except Exception as e:  
 raise e  
 return a, b, c

## Результат выполнения программы

The solutions are: {2.0, 3.0, -3.0, -2.0}