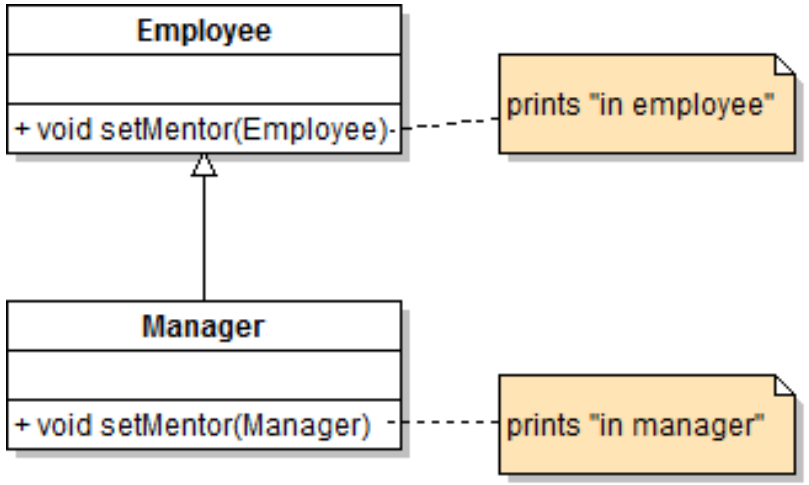
Today

1. Recap on method dispatch and inheritance
2. Basics of the Exception-Handling Mechanism
3. Exception Handling Patterns
4. Encapsulation and Exception Handling

About Mentoring

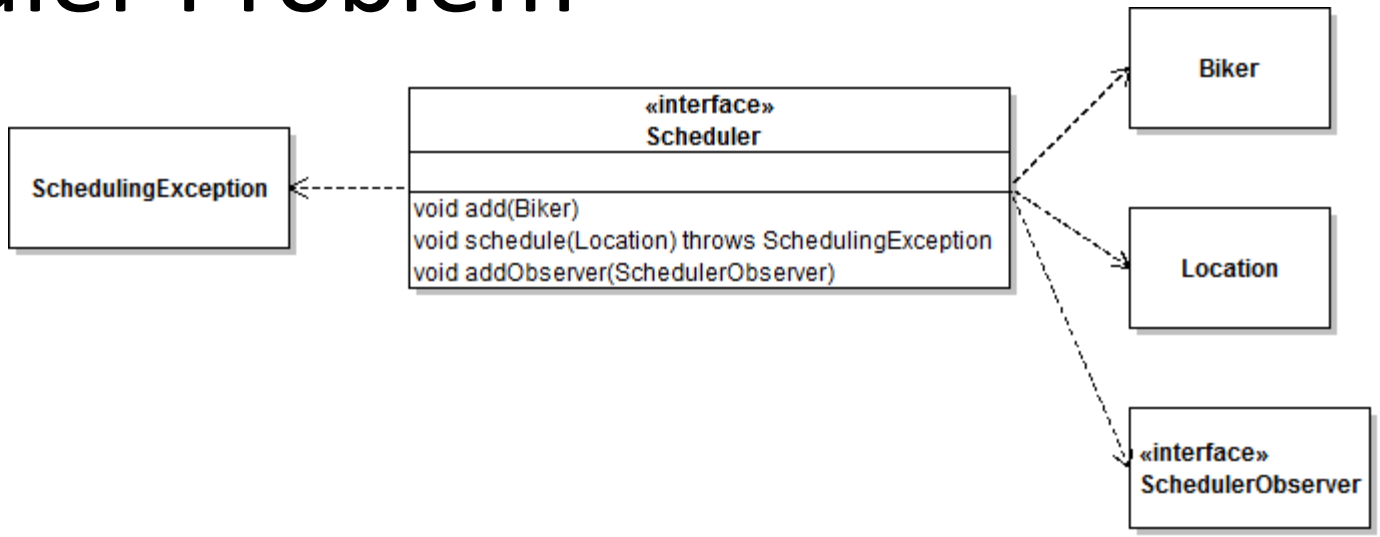
(Exercise Set 07)




```
public static void main(String[] args)
{
    Employee alice = new Employee("Alice");
    Employee bob = new Employee("Bob");
    Employee carl = new Manager("Carl");
    Employee diane = new Manager("Diane");

    alice.setMentor(bob);
    carl.setMentor(diane);
```

Scheduler Problem

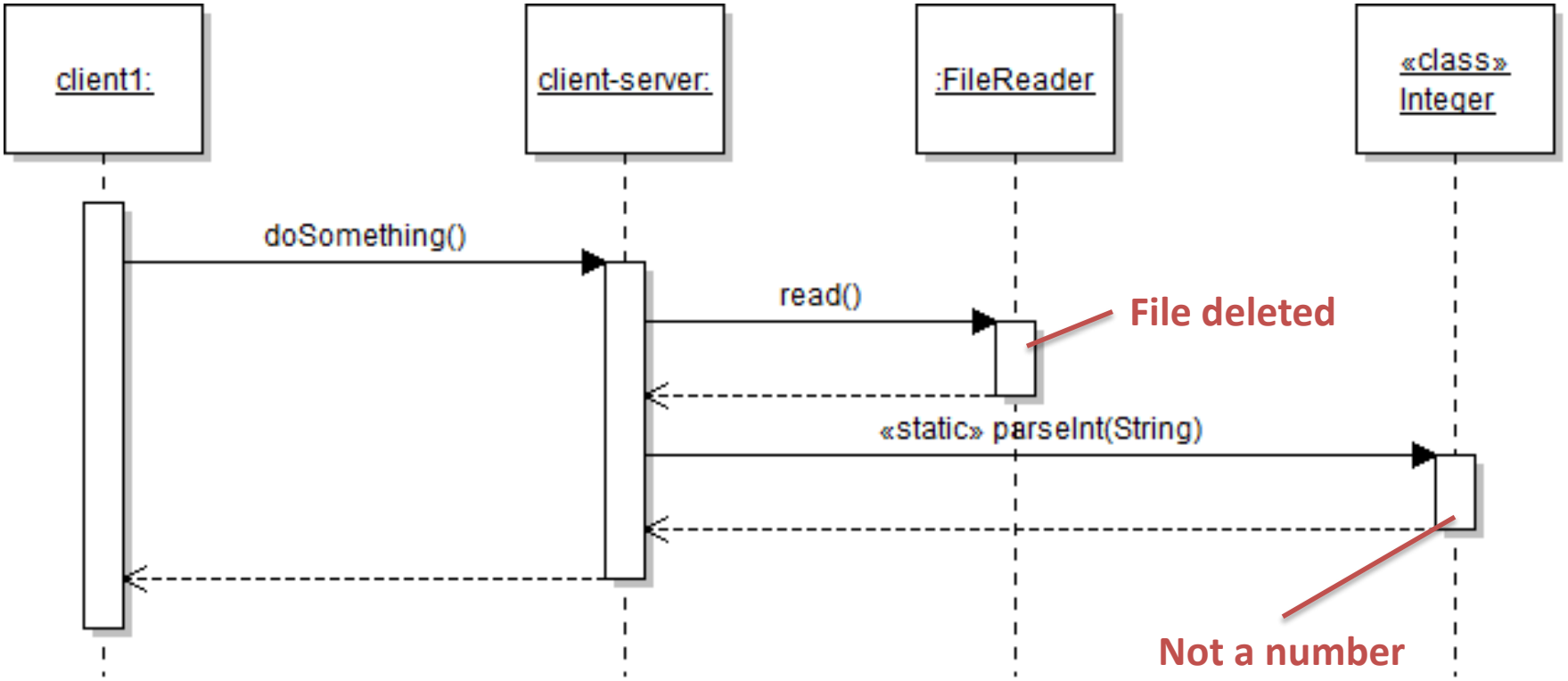


A bike courier company uses a Scheduler system to schedule bikers for delivery based on various factors (unimportant for this practice question). The company wants the flexibility to install different scheduling algorithms. However, all scheduling algorithms should follow these steps:

a) check if at least one biker is available, and if not throw an exception;
b) schedule a biker using a given algorithm;
c) notify interested observers that a biker was scheduled.

Operations a) and c) are the same for all algorithms, but should be isolated in separate methods. Concrete schedulers should also have the flexibility to throw algorithm-specific types of exceptions if they cannot fulfill a scheduling request. Assume all exceptions for this design are checked. Complete the following UML class diagram to provide a design for these requirements. Use the TEMPLATE METHOD design pattern. When relevant to the design, make sure to include the appropriate modifiers for methods and/or classes (final, public, protected, private, abstract, etc.).
Illustrate support for two different scheduling algorithms. Include the OBSERVER mechanism for biker notification. Write the code necessary to implement the relevant parts of your design.

Exception Handling: Ground Zero

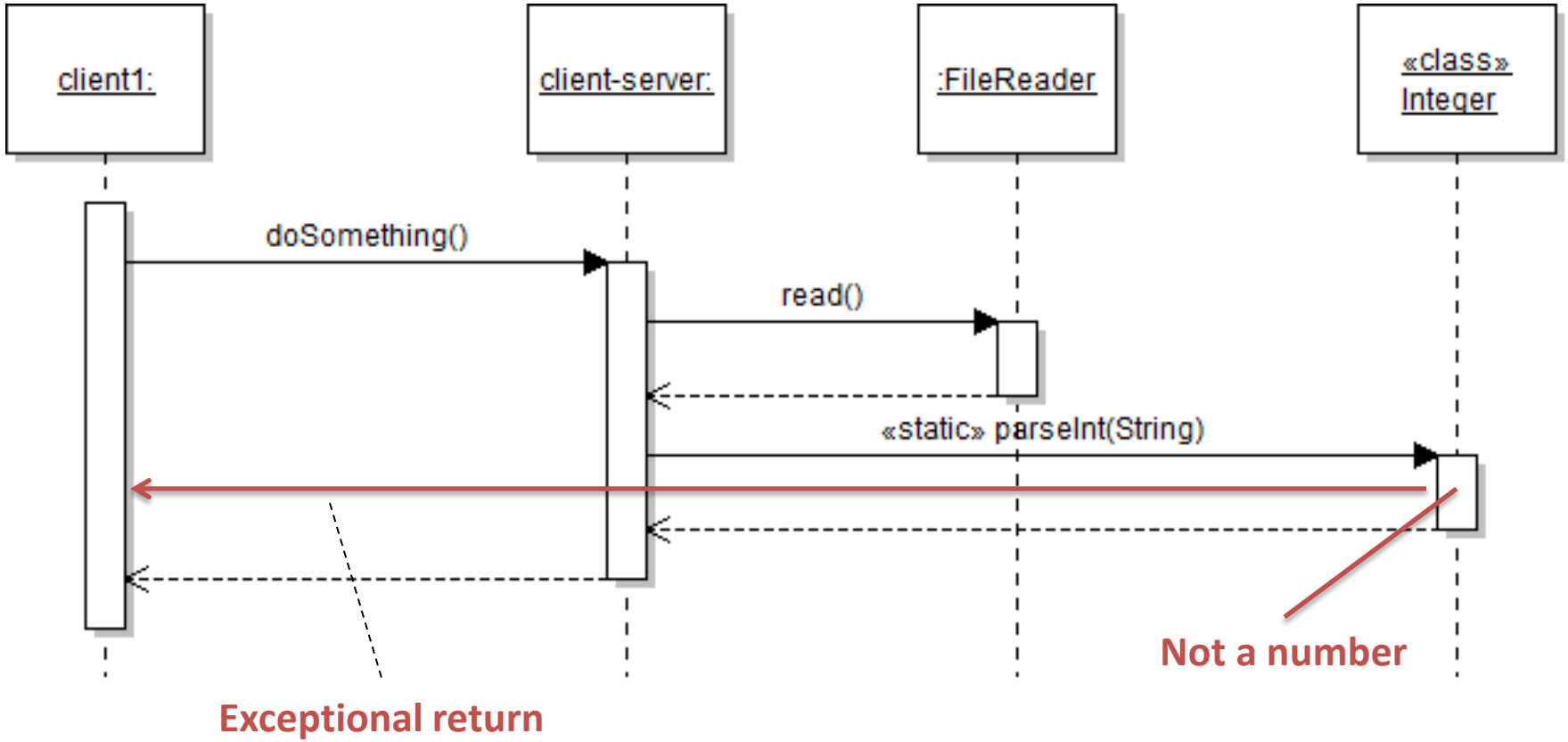


Some dusty C code

```
FILE *stream;
Stream = fopen(name, “r”);
if(!stream) {
   fprintf(stderr,…);
   return IO_FAILURE;
   …
```

Problems with that:

Exception Handling: Ground Zero



Exception handling allows you to deal with a problem at a location different from where you detect it. Exceptions propagate though the call stack.

Basic Observer Design