Lecture 8

- Kernel and Address Spaces: Security in Java
- Address Translation
- Hw 3: Strategy pattern, specializing algorithms

Class SecurityManager

- The current SecurityManager:
  - System.getSecurityManager()
- Contains a large number of methods whose name begins with “check”.
- Called by various methods throughout Java libraries before those methods perform sensitive operation.

Invocation of check

```
SecurityManager security =
  System.getSecurityManager();
if (security!= null) {
   security.checkXXX(arguments);
}
```

may throw `SecurityException`

java.lang.SecurityManager

```
public abstract
  class SecurityManager {
  protected boolean inCheck;
  public void checkAccess(Thread t)
    throws SecurityException;
  public void checkExit(int status)
    throws SecurityException;
  public void checkExec(String cmd)
    throws SecurityException; …}
```

Protection in Java

- java.security
  - contains tools for security related functions.
  - Digital signatures
  - Access control lists
  - Many ways to do cryptography: provides general interface

Hw 3

Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way.
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithm for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

Very interesting problem: parameterization of algorithms

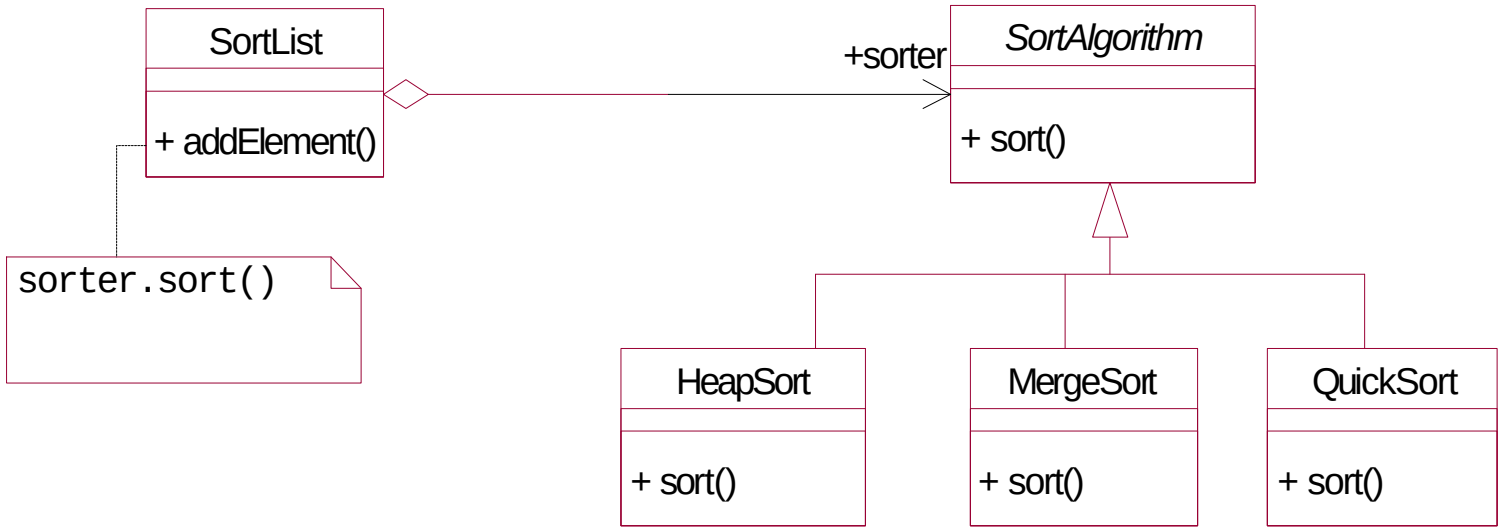
- Not only of interest in Operating Systems but in software development in general.
- Large business systems: consist of basic business processes that are specialized in different ways
- Pricing algorithm: Negotiated, Aging, Frequent, Discount

Behavioral Patterns

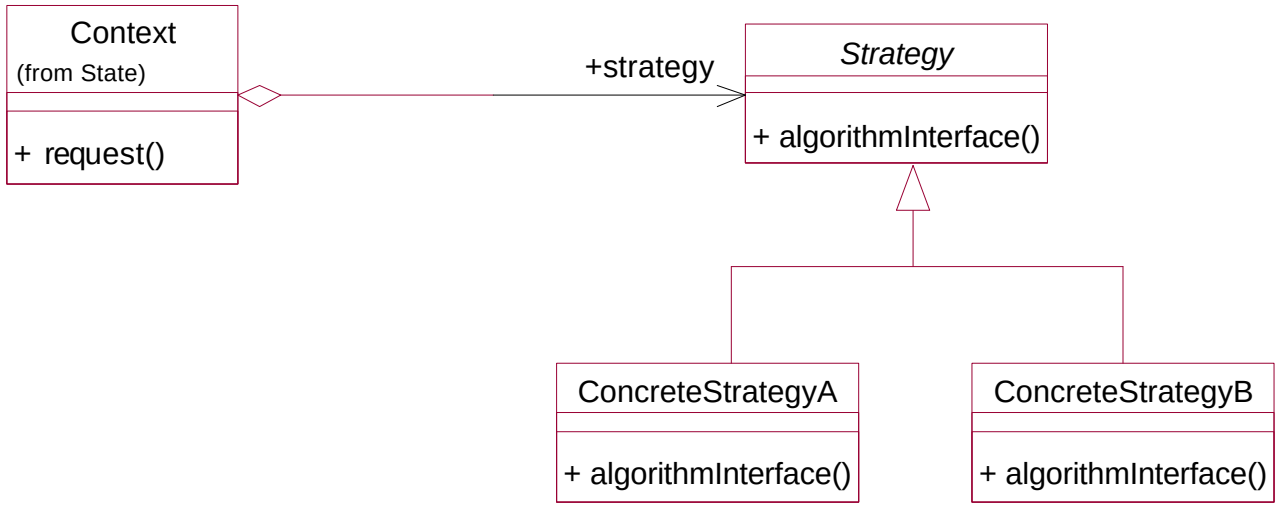
Strategy - Intent

Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from the clients that use it.

Strategy - Example



Strategy - Structure



Strategy - Applicability

- When many related classes differ only in their behavior. Strategies provide a way to configure a class with one of the many behaviors.
- When you need different variants of an algorithm.
- To avoid exposing complex, algorithm-specific data structures.
- To move partitions of a conditional statement into its classes

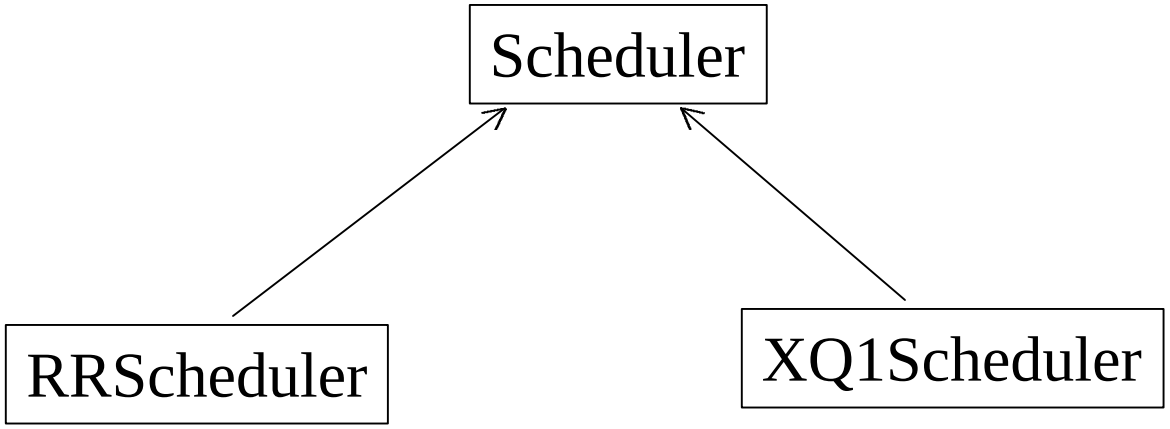
Strategy - Consequences

- ✓ Hierarchy of Strategy classes defines a family of algorithms or behaviors for Context to reuse.
- ✓ An alternate to subclassing. Encapsulating the behavior in separate Strategy class lets you vary the algorithm independently of its Context, making it easier to switch, understand and extend.
- ✓ Eliminates conditional statements.

Strategy - Consequences

- Clients must be aware of different Strategies.
- Communication overhead between Strategy and Context.
- Increased number of objects. Flyweight can be used to share Strategies.

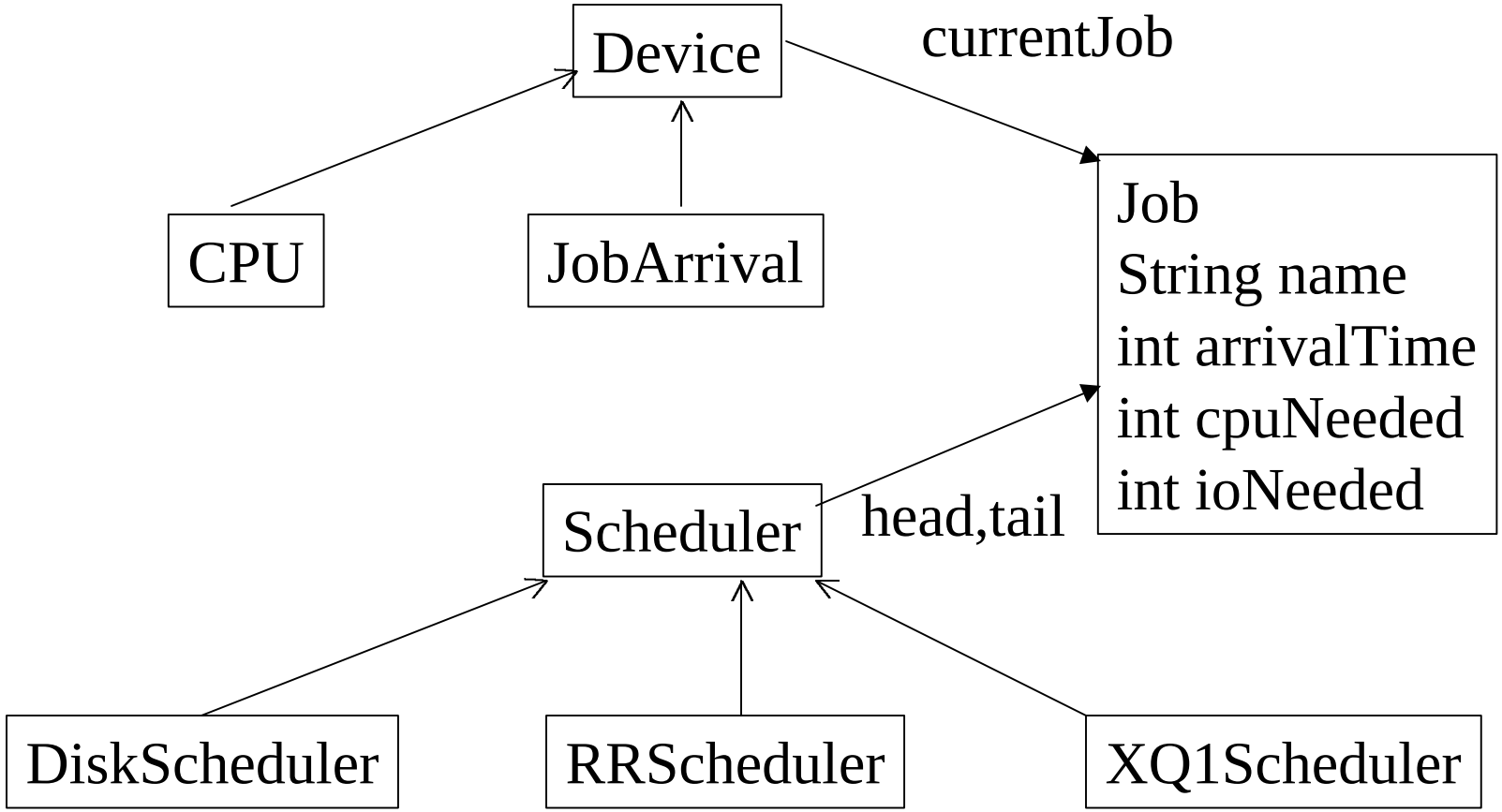
How to improve program?



Pass Scheduler-objects around!

Rule to remember

- Addition is good, modification is bad.
  - Adding a new class is good, modifying conditional statements is bad.
  - Adding a new scheduler involves adding a new subclass.
- Keep information about one issue in one place. Information about RoundRobin belongs to `RRScheduler`


Device
currentJob
CPU
JobArrival
Job
String name
int arrivalTime
int cpuNeeded
int ioNeeded
Scheduler
head,tail
DiskScheduler
RRScheduler
XQ1Scheduler