Polymorphism

Lab 6

TA : Changmin Jeon, Minji Kim, Hyunwoo Jung,
Hyunseok Oh, Jingyu Lee, Seungwoo Jo

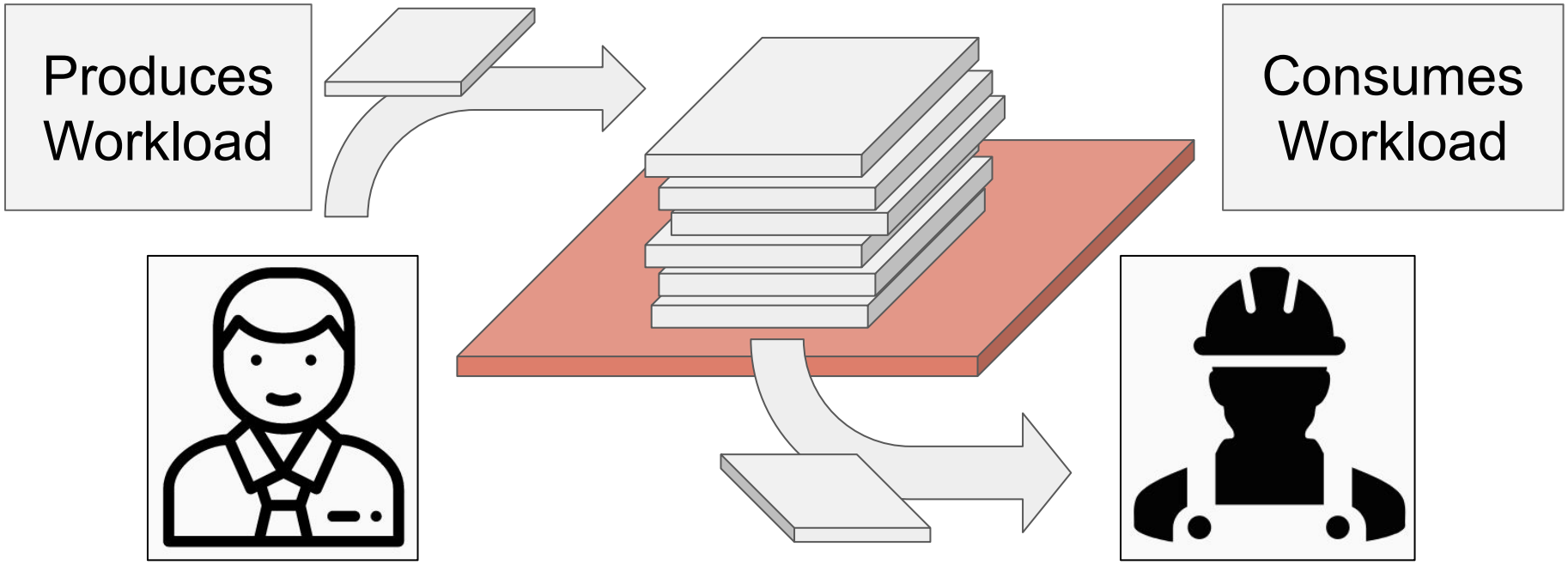
SEOUL NATIONAL UNIVERSITY

Objectives

- Understand the differences of static and instance members, and observe the practical usage of these differences.
- Understand the advantages of Overriding in java.
- Understand the Generics and its advantage.

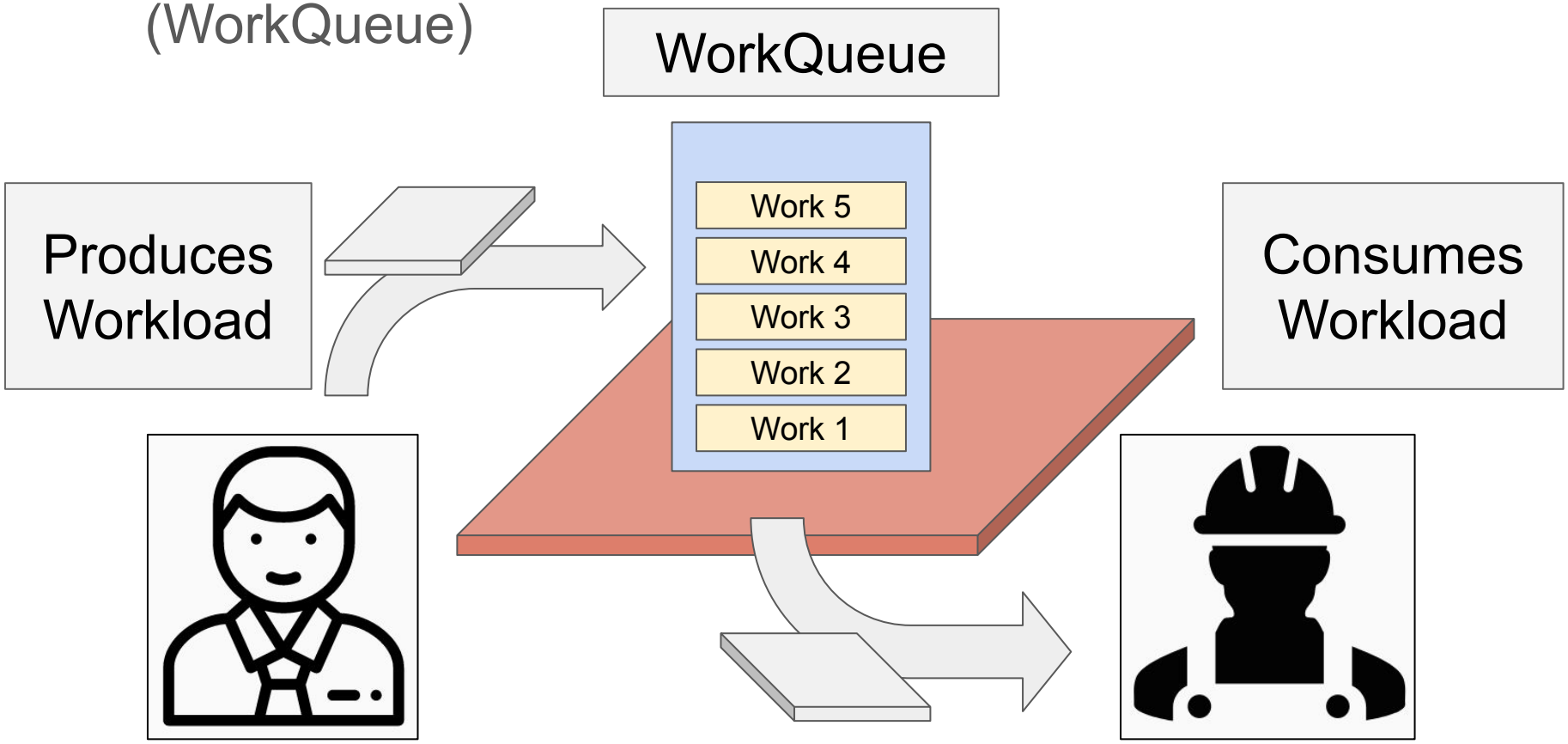
Producer-Consumer Pattern

- Workload(Pile of works to be done) exists.
- *Producer* produces works and pile up in the workload.
- *Consumer* consumes works from the workload.



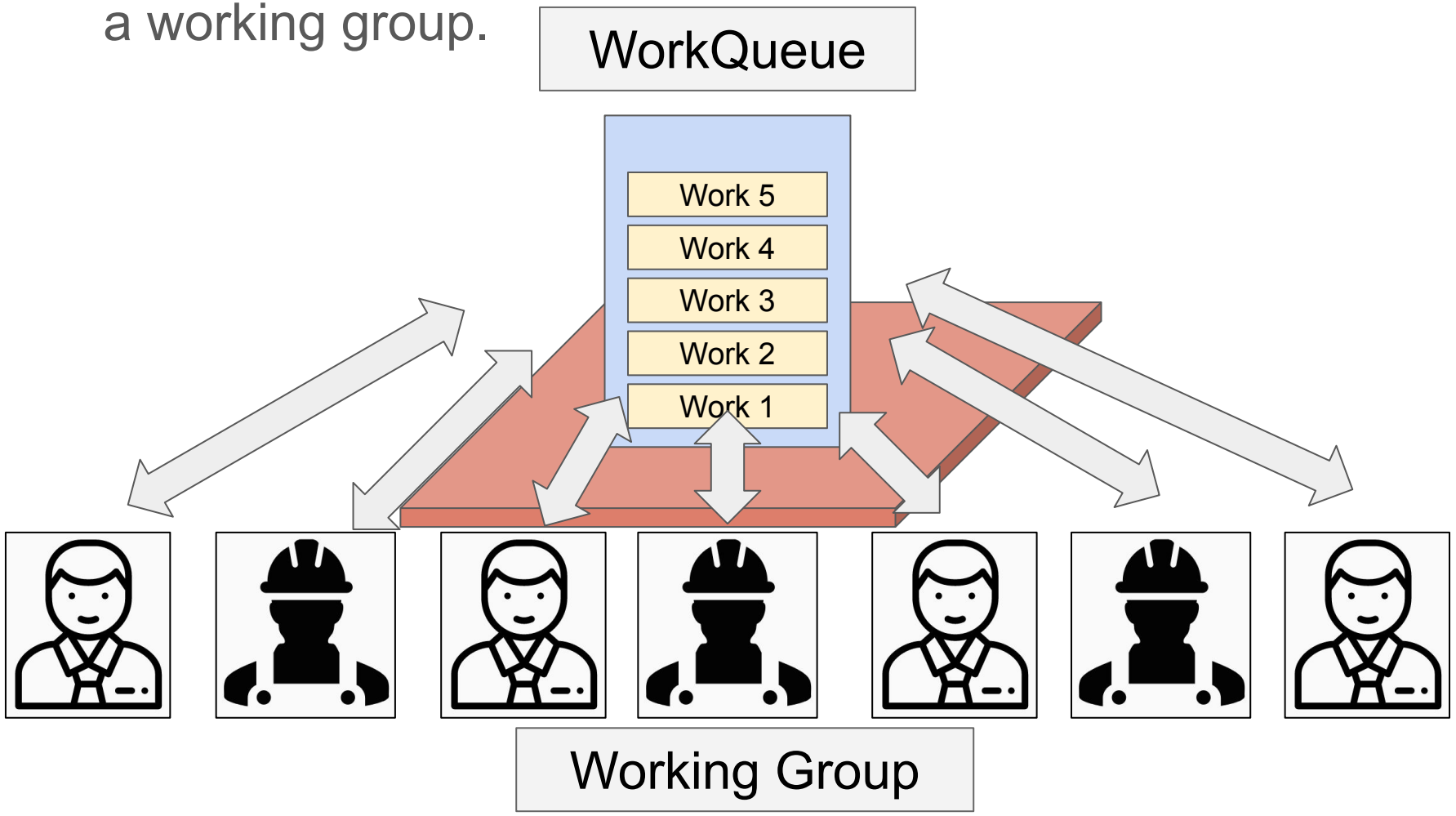
Producer-Consumer Pattern

- Workload could be represented as a Queue of Works (WorkQueue)



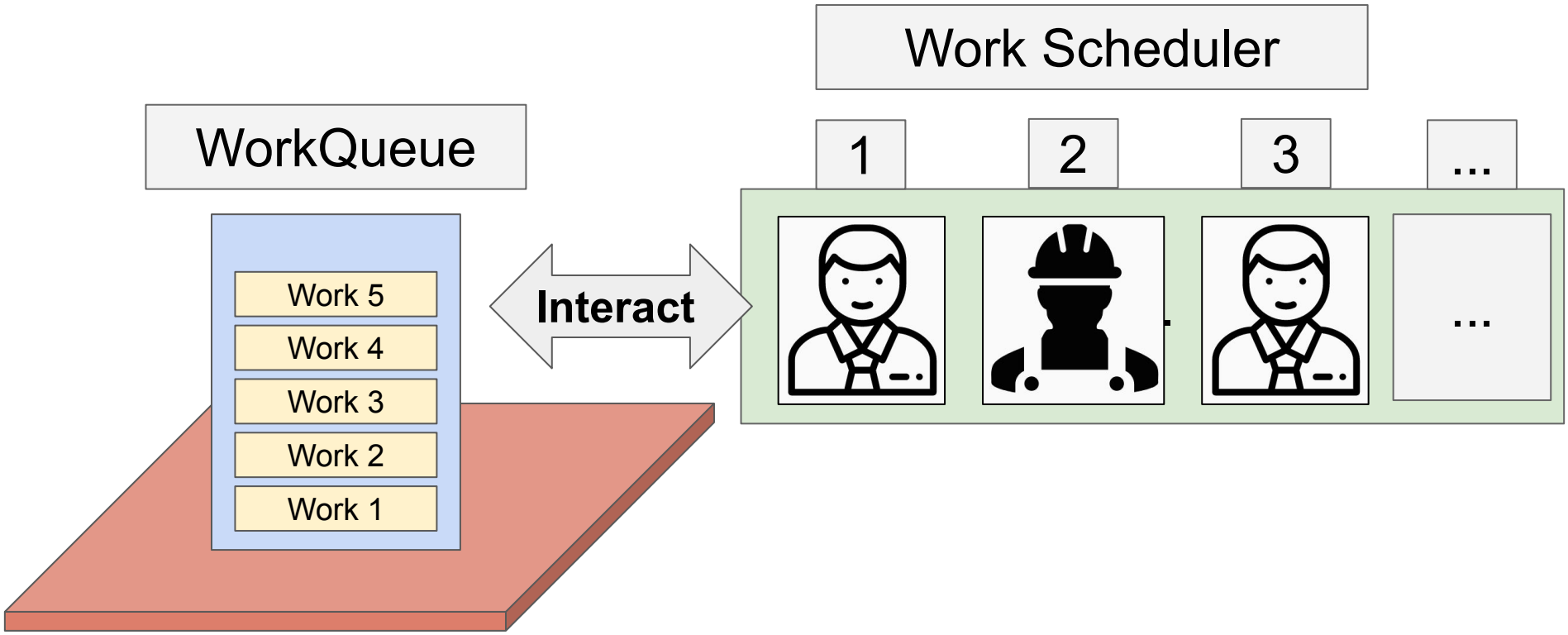
Producer-Consumer Pattern

- Several Producers and Consumers could exist. They form a working group.



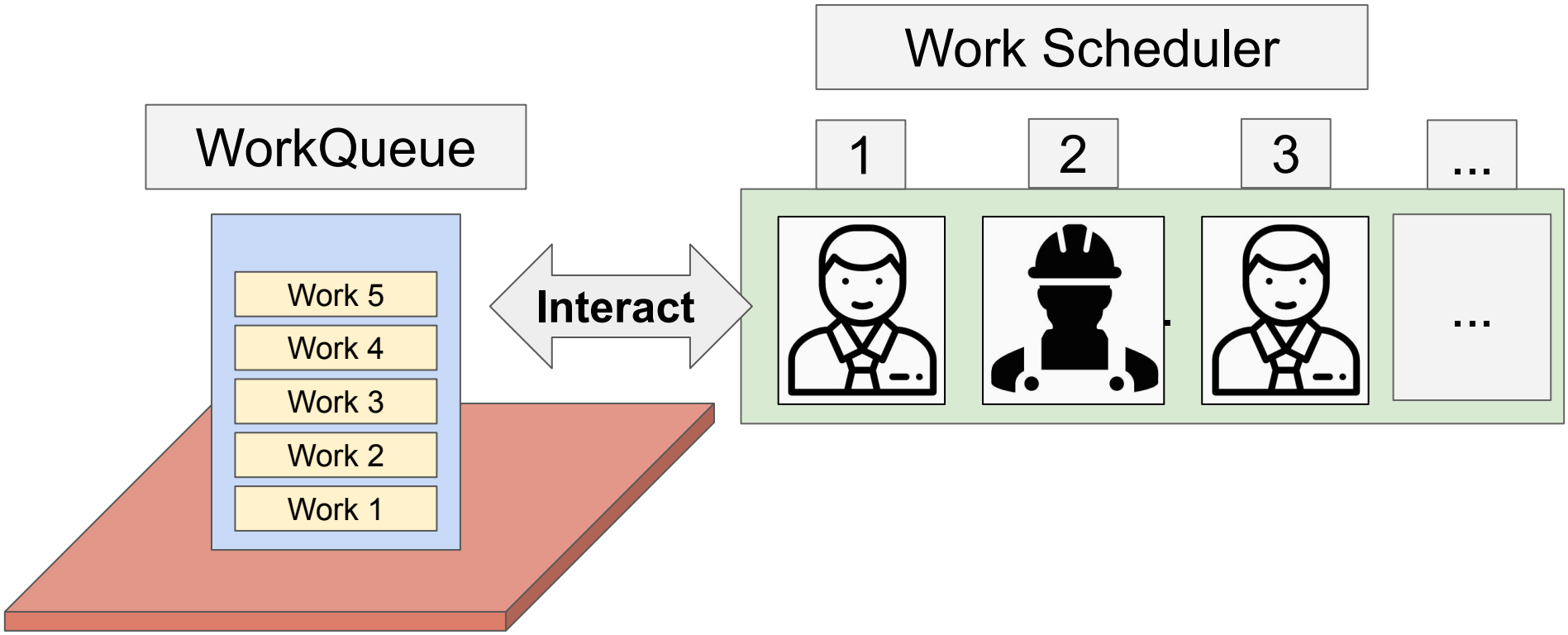
Producer-Consumer Pattern

- Scheduler gives one of them the opportunity to do their work for a given period of time. (scheduling)
- The policy of selecting the worker could be very diverse.



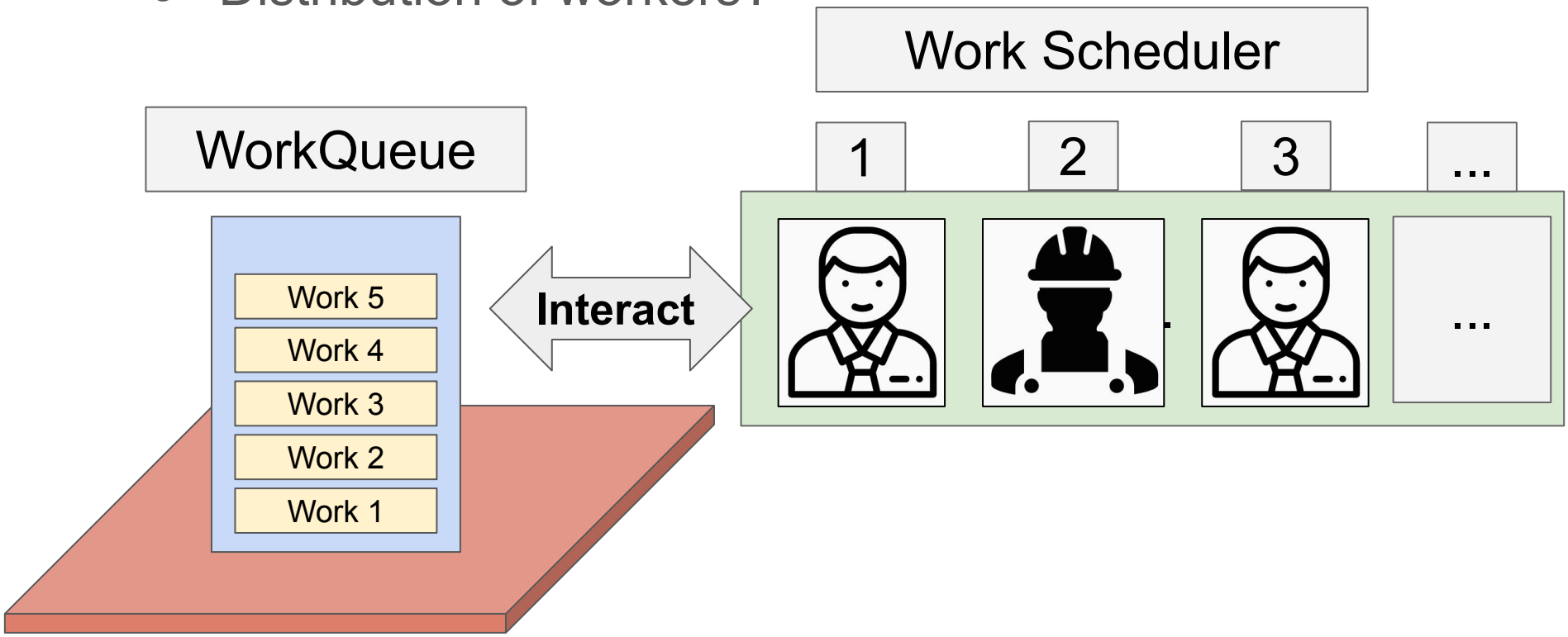
Producer-Consumer Pattern

- Both producer and consumer being a class, and a workqueue being a computational workload, it is called a producer-consumer pattern.



Producer-Consumer Pattern

- What kind of characteristic of workqueue will be observed as we differentiate
  - the policy of this scheduling?
  - Distribution of workers?



Overview

- Problem1 - Work ID and Worker ID
  - Managing WorkQueue
  - Managing Workers
- Problem2 - Producer and Consumer
  - Worker class
- Problem3 - Scheduling

Managing WorkQueue

- Initialize with Works
- Workers will insert and pop works.
- Queue API
  - Insert : `workQueue.add(new Work());`
  - Take out : `Work work = workQueue.poll();`

```
Queue<Work> workQueue;
workQueue = new LinkedList<>();
for(int i = 0; i < numWorks; i++){
    workQueue.add(new Work());
}
```

Managing Workers

- Working Group for managing the workers, each worker of type `Producer` or `Consumer`.
  - `List<Worker>` : List of Workers
- Working Group contains derived classes of the Worker.

```
List<Worker> workers;
workers = new LinkedList<>();
for (int i = 0; i < numProducers; i++) {
    workers.add(new Producer(workQueue));
}
for (int i = 0; i < numConsumers; i++) {
    workers.add(new Consumer(workQueue));
}
```

Problem1 - Work ID and Worker ID

- Add id attribute to `Work` and `Worker` class
- Consider static attribute.
- Every `Work` have their own id.
  - The first `Work` object id is 0, the second `Work` object id is 1, ...
- Every `Worker` object (including `Producer` and `Consumer`) have their own id.

Worker class

```
public abstract class Worker {
    public abstract void run();
    Worker(Queue<Work> workQueue) {
        this.workQueue = workQueue;
    }
}
class Producer extends Worker {
    Producer(Queue<Work> workQueue) { super(workQueue); }
    public void run() { }
}
class Consumer extends Worker {
    Consumer(Queue<Work> workQueue) { super(workQueue); }
    public void run() { }
}
```

Problem2 - Producer and Consumer

- Implement `public void run()` method of `Producer` and `Consumer` class.
- The `run` method of the `Producer` class always adds a `Work` object to the workQueue.
- The `run` method of the `Consumer` class probabilistically (⅓) takes out a `Work` object to the workQueue.

Scheduler

`Scheduler` class defined under unknown type variable `T`.

- `Scheduler<Worker>` : worker group scheduler
- On `schedule()`, it samples one of the `T` from the group with its internal policy and returns it.
- Scheduler need not know further knowledge about the input type `T`.

```
public class Scheduler {
    Worker schedule(List<Worker> workers) {
        // Add schedule logic
        return workers.size() == 0 ? null : workers.get(0);
    }
}
```

Problem3 - Scheduling

- Implement `T schedule(List<T> workers)` method in `Scheduler` class.
  - Return a randomly selected worker among input workers.
- Add report message to the `public void run()` method of `Producer` and `Consumer` class.
  - Use the `report()` method in `Worker` class.
  - When the producer produces a work
    - report(“produced work<Work ID>”)
  - When the consumer consumes a work
    - report(“consumed work<Work ID>”)
  - When the consumer fails to consume a work. (2/3 probability failure or no work in the workqueue)
    - report(“failed to consume work”)

Submission

- Compress your `src` directory into a zip file.
  - After unzip, the ‘src’ directory must appear.
- Rename your zip file as 20XX-XXXXX_{name}.zip - for example, 2020-12345_KimMinji.zip
- Upload it to eTL - Lab 6 assignment.