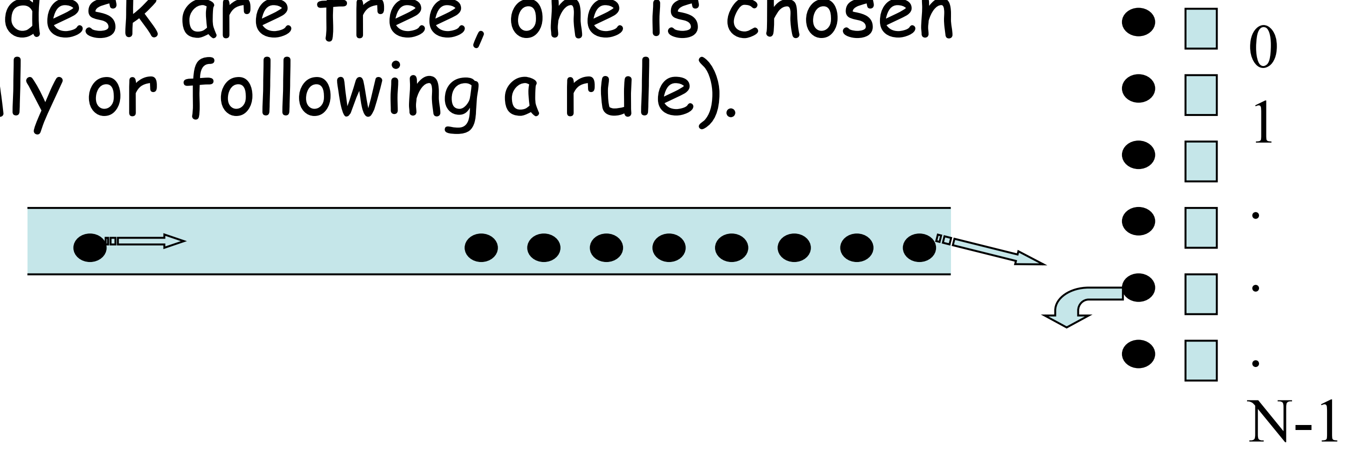
Sportelli - 1

We want to implement the control system of a waiting queue for a service made of N desks able to service one customer at a time.

Each service has a random duration within a range of times.

When a service ends, another customer, if waiting, can access the freed desk. If a customer comes when more than one desk are free, one is chosen (either randomly or following a rule).



Analysis

Read carefully the text, recognizing the important requirements that must be thoroughly respected

1. Define **shared variables** (buffers, 'pointers', state and counting variables, etc.)
2. Identify **synchronization conditions**
3. Insert required synchronizations within the **requested methods** in order to fulfill the requirements

If you define **threads as inner classes** in the application class, **shared variables** and synchronization methods, defined as (even private) elements of the containing class, are **accessible** to the inner class methods.

Shared variables

- **Count** the number of **customers** in the queue
- **Represent** the state of the system (which desks are free)
- **Impose** mutual exclusion when necessary (how to do it depends on the used synchronization tool)
- Other variables depending on the used synchronization tool

Synchronization conditions

- **A customer must wait**
  - If she is not at the head of the queue
  - If there is no free desk
- **When at least one desk is free (or becomes free)**
  - The customer on the head of the queue chooses a free desk for service
  - The service has a random duration

Synchronized methods

We call the application class **Sportelli*Type*** with ***Type*=**
(Sem,Reg,Mon,Jav) according to the synchronization tool. It includes these synchronization methods:

1. **int entraCoda()** where a customer may be forced to wait
2. **int esce(int sport)** the customer frees the desk *sport* in favor of a waiting customer (or future customer if the queue is currently empty)

Thread - 1

- The customer is represented by a thread instance of the **ClienteTh** class which extends the **Thread** class (it does not have to extend another class)
- This class is defined as a **inner member class** of **SportelliXX** so that it can access the shared variables and methods in the associated instance of **SportelliXX**
- Its **run()** method executes the actions of a customer
- The **main** thread (the one executing the **main** method) creates the necessary instances and activates application threads

Development phases

1. **Petri net** optional
2. **main() method**
   - Creation of instances
   - Thread activations
3. **Thread classes**
   - Constructor
   - run() method
4. **Synchronization class**
   - Synchronization tools

Semaphore (binary) in Java

```
class Semaphore {
    public Semaphore(boolean b);  // b initial value

    public synchronized void p();  // atomic operation p

    public synchronized void v();  // atomic operation v

    public synchronized long p(long timeout);
                                  // atomic op. p with timeout (ms)

    public int value();           // semaphore value (0 o 1)

    public int queue();           // # of enqueued threads

    public synchronized Thread waitingThread(int pos);
                                  // waiting thread at position pos

    public String toString();     // descriptive string
}
```

At work

. . . .

The end

05.c

Sportelli

Semaphores, Regions, Monitors