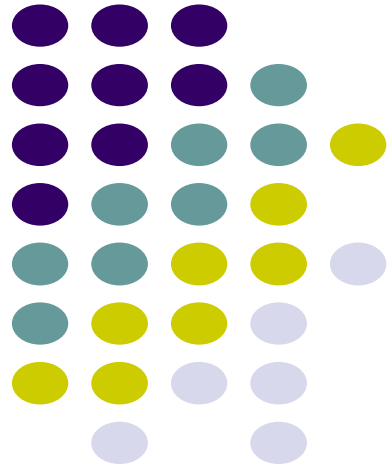
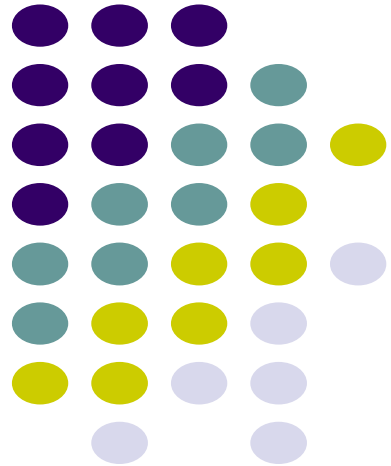
# Course
# Parallel Programming and Distributed Systems in Java

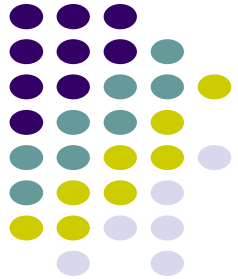


# Лекция 2

# Взаимодействие потоков

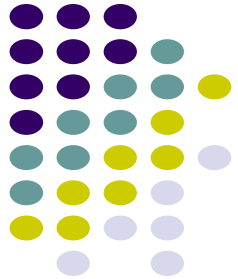


# Когда нужно, чтобы потоки взаимодействовали?



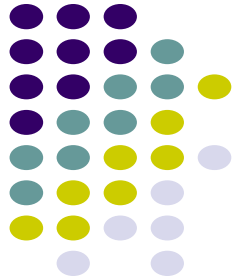
- Доступ к критическим блокам (Mutual Execution)
- Обмен данными (Coordination)

# Mutual exclusion



- Как «защитить» критичный ресурс?
- Lock?
- Что, если критичных ресурсов несколько?
- Mutual Exclusion – потенциальный bottleneck

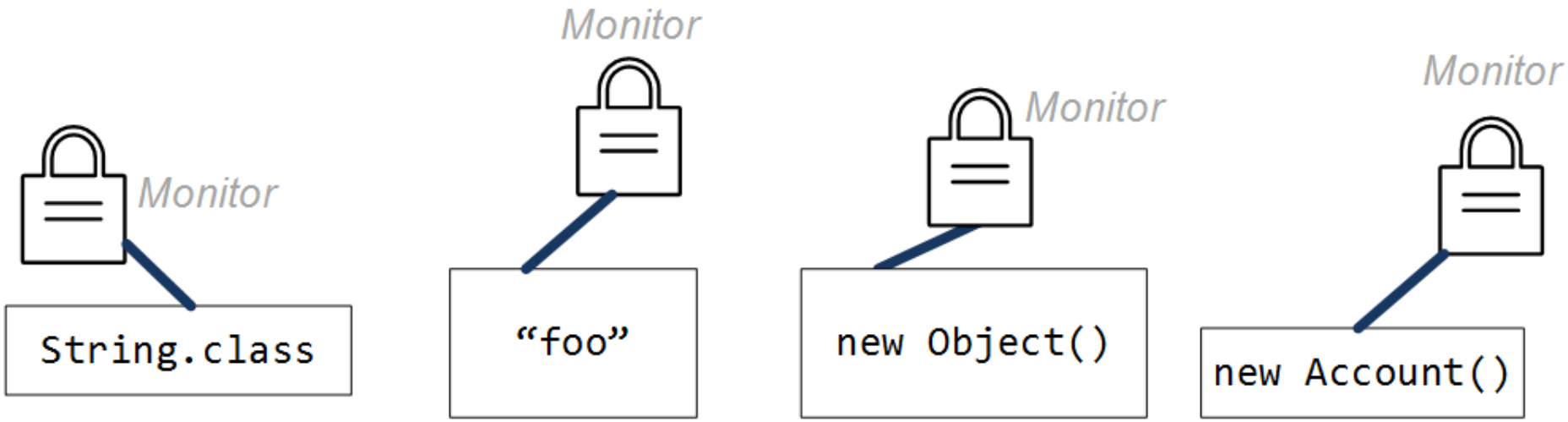
# Producer Consumer



- Обмен сообщениями между потоками

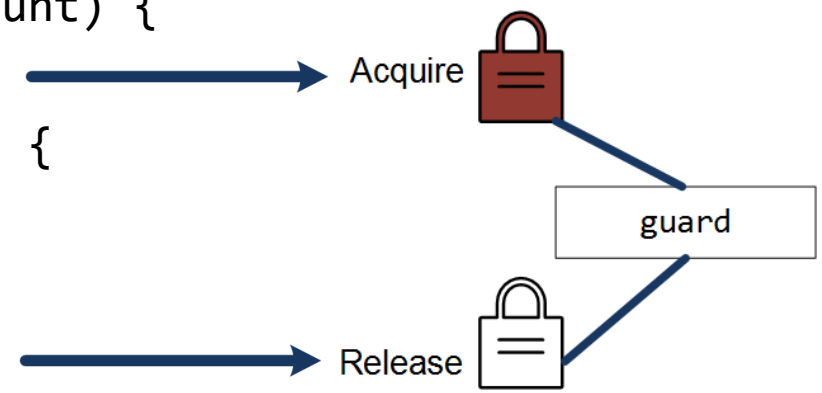
# Механизмы Java

- Монитор – у каждого объекта в Java есть свой монитор

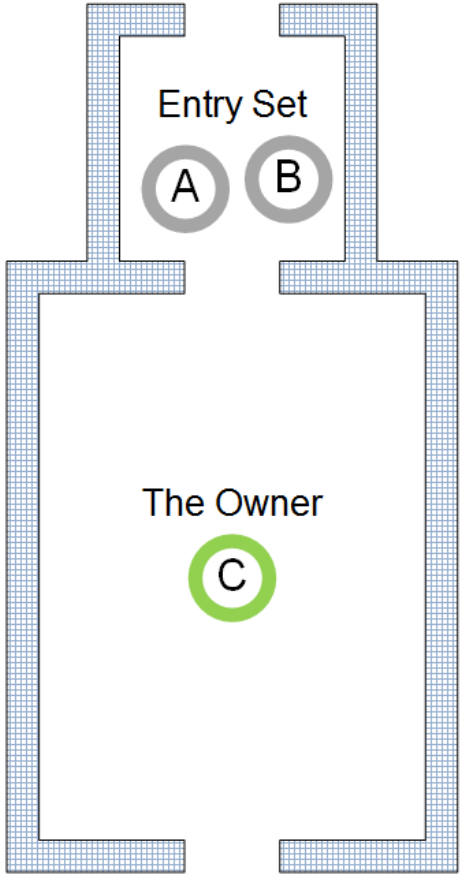


# Code

```
public class Account {

    private int total = 0;
    private Object guard = new Object();

    public void withdraw(int amount) {
        synchronized (guard) {
            if (total >= amount) {
                total -= amount;
            }
        }
    }
}
```



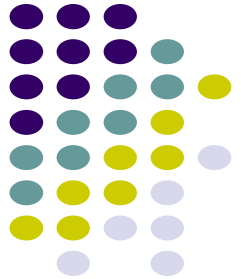
# How synchronized works



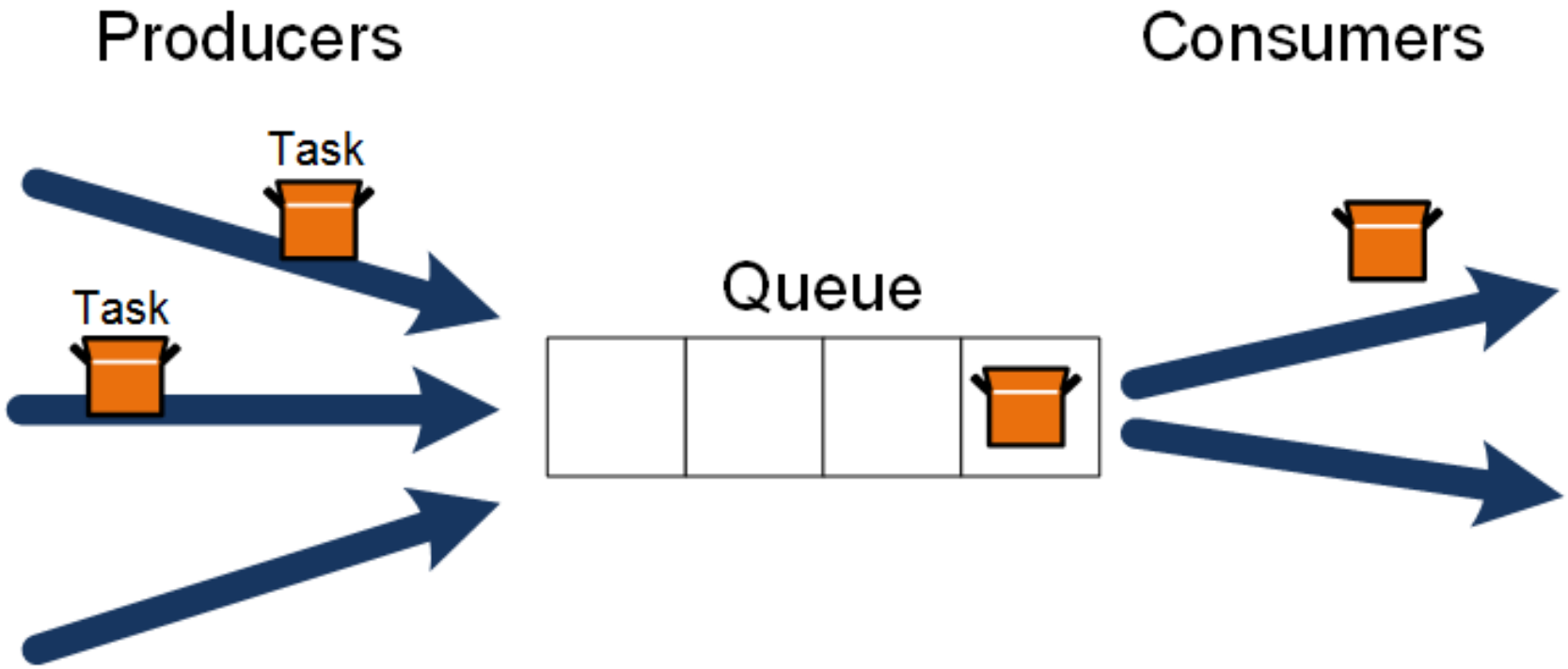

```
synchronized (guard) {

      if (total >= amount) {
        total -= amount;
      }

}
```

# More code

```
public synchronized void withdraw()

public static synchronized void withdraw()

synchronized (guard) {
        ...
}

synchronized (this) {
        ...
}
```



# Producer Consumer



# wait / notify

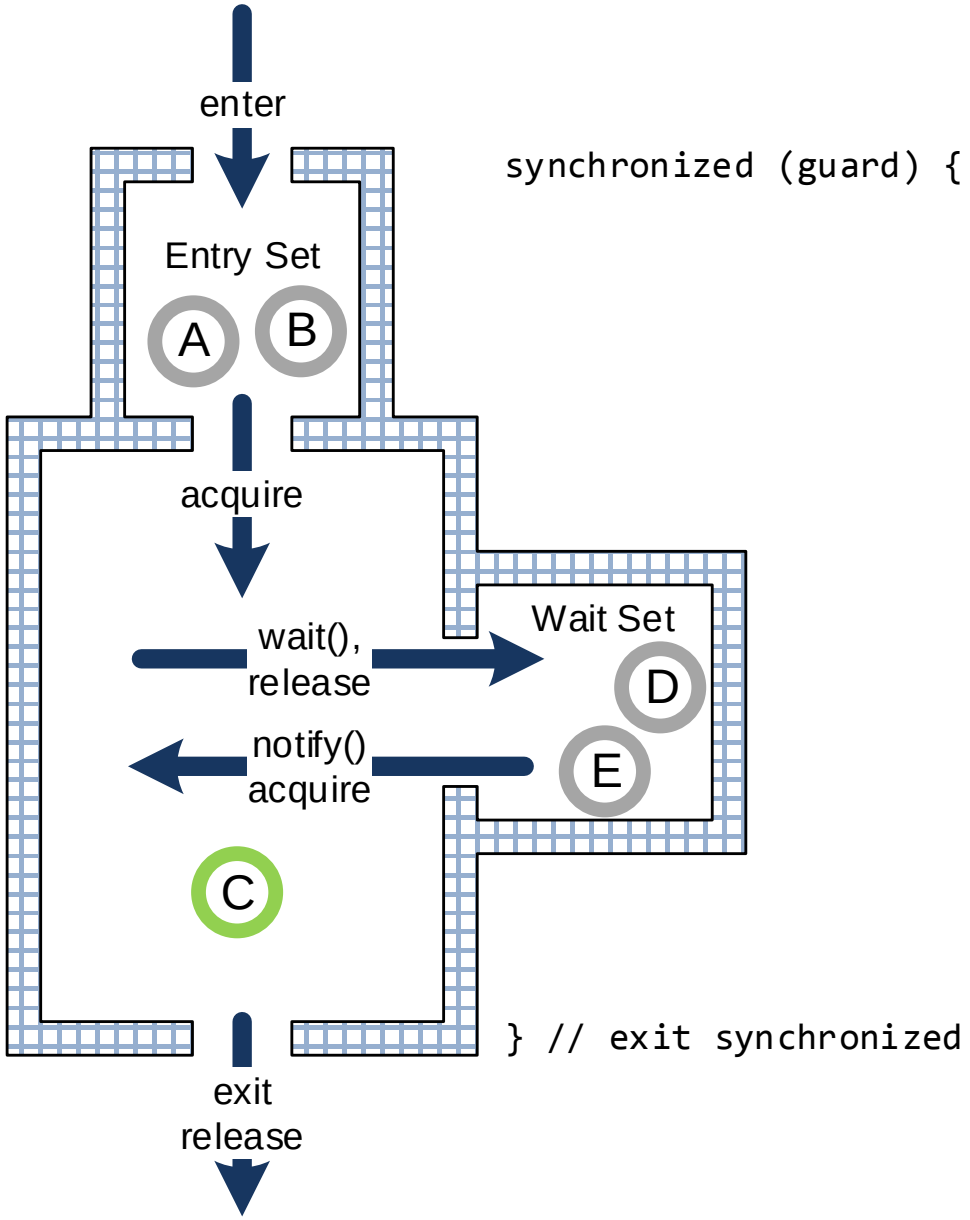
Thread A

```
synchronized(guard) {
  try {
    guard.wait();
  } catch(InterruptedException e) {
}
```

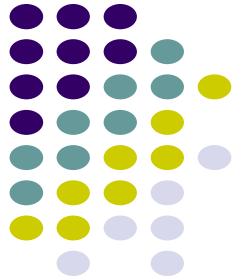
Thread B

```
public void notify() {
  synchronized(guard){
    guard.notify();
  }
}
```

# The big picture

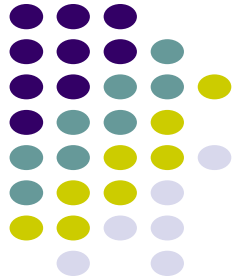


# java.util.concurrent



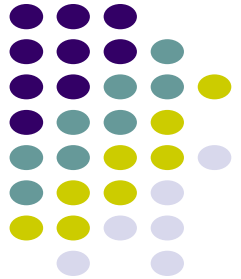
- Atomics
- Locks / Conditions
- Coordination classes
- Thread pool / Executor service
- Queues

# Synchronized = Lock



```
private final Lock lock = new ReentrantLock();
// ...

public void withdraw() {
  lock.lock();
  try {
    // ... method body
  } finally {
    lock.unlock()
  }
}
```

# wait / notify = Condition



```
private final Lock lock = new ReentrantLock();
private final Condition condition =
                        lock.newCondition();

public void waitForJob() {
  lock.lock();
  try {
    condition.await(); // like wait()
  } finally {
    lock.unlock();
  }
}
```

```
public void addJob() {
  lock.lock();
    try {
      condition.signalAll(); // like notifyAll()
    } finally {
      lock.unlock();
    }
  }
}
```