Threads
Threads

- Concurrent executions
- Separate stacks
- Supported by the JVM
- You should know
Creation

subclass **Thread**, and override **run**.

E.g., a thread that computes primes larger than a stated value [1]:

class PrimeThread extends Thread {

    long minPrime;
    PrimeThread( long minPrime ) {
        this.minPrime = minPrime;
    }

    public void run() {
        // compute primes larger than minPrime  . . .
    }
}

Create a thread and start it running:

PrimeThread p = new PrimeThread(143);
p.start();
Creation, cont.

declare a class that implements the **Runnable interface**; then implements the `run` method. An instance of the class can then be allocated, passed as an argument when creating Thread, and started [1]. E.g.:

class PrimeRun implements Runnable {
    long minPrime;
    PrimeRun(long minPrime) {
        this.minPrime = minPrime;
    }
    public void run() {
        // compute primes larger than minPrime . . .
    }
}

create a thread and start it running:
PrimeRun p = new PrimeRun(143);
new Thread(p).start();
Synchronization

- Critical section: a region of code that at most one thread can execute at a time
- Every object has a lock
- Can synchronize methods [2]:

```java
class Account {
    private double balance;
    public Account( double initialDeposit ) {
        balance = initialDeposit;
    }
    public synchronized double getBalance() {
        return balance;
    }
    public synchronized void deposit( double amount ) {
        balance += amount;
    }
}
```
Synchronization, cont.

- Static synchronized methods acquire the lock of the Class object for their class.

- `synchronized` statements
```java
synchronized( anObject ) {
    statements
}
```
Synchronization, cont.

• Beware of deadlock!
• Deadlock $\iff$ cycles in the waits-for graph
Refs:
