M9(a)-Concurrency

Jin L.C. Guo

Image source: https://cdn.pixabay.com/photo/2016/10/10/00/48/chefs-1727446_960_720.jpg

Objective

- Understand the concept of a Thread and its usefulness for programming;
- Be able to write basic concurrent programs in Java;
- Understand the causes of basic concurrency errors
- Understand the mechanisms that help prevent the basic concurrency errors.

Make this ravioli dish

- Make the pasta dough
- Let the dough rest
- Make the filling
- Make the ravioli
- Boil the water
- Cook ravioli in water
- Make the butter sauce
- Finish cooking ravioli in sauce
- Make garnishing

Source: https://www.jamieoliver.com/recipes/pasta-recipes/amazing-ravioli/

Units of execution

- Process
  - A self-contained execution environment
  - Has its own memory space
  - Most implementations of Java virtual machine run as a single process

Units of execution

- Thread
  - Lightweight process
  - One process has at least one thread.
  - Threads in one process share process resources
    - Memory
    - File handles
    - Security credentials
  - Thread has their own
    - Program counter
    - Stack
    - Local variables
  - Each java application start with one thread: main thread.

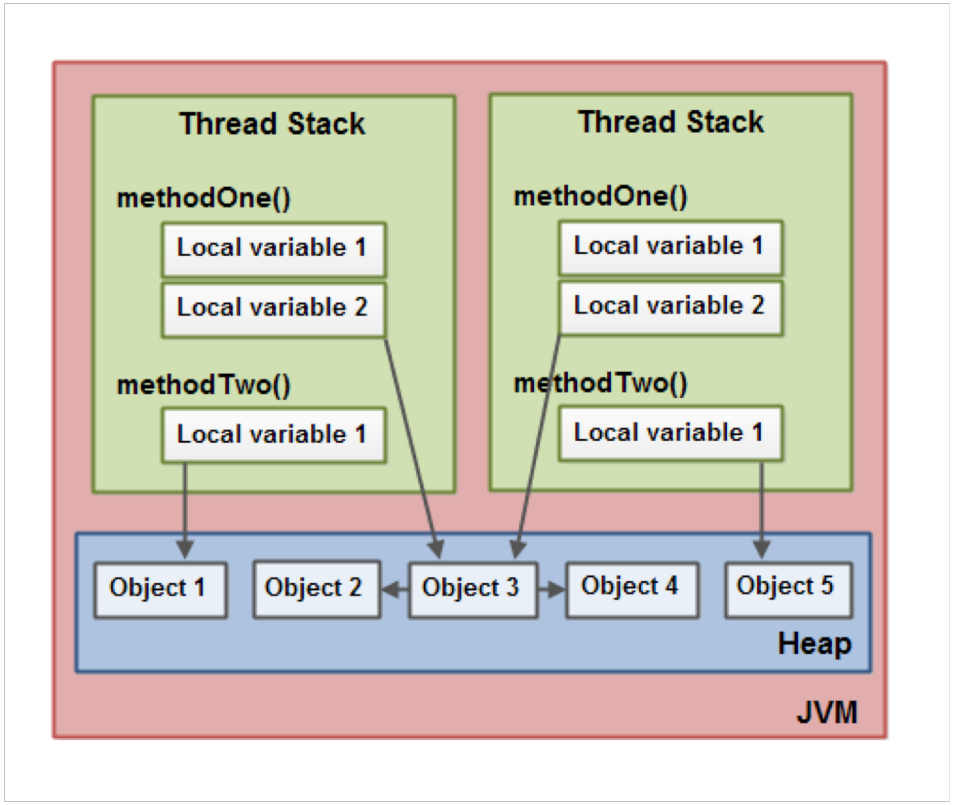
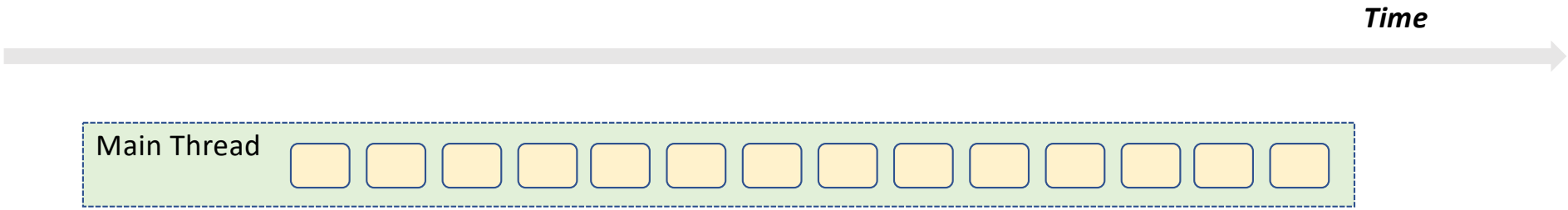


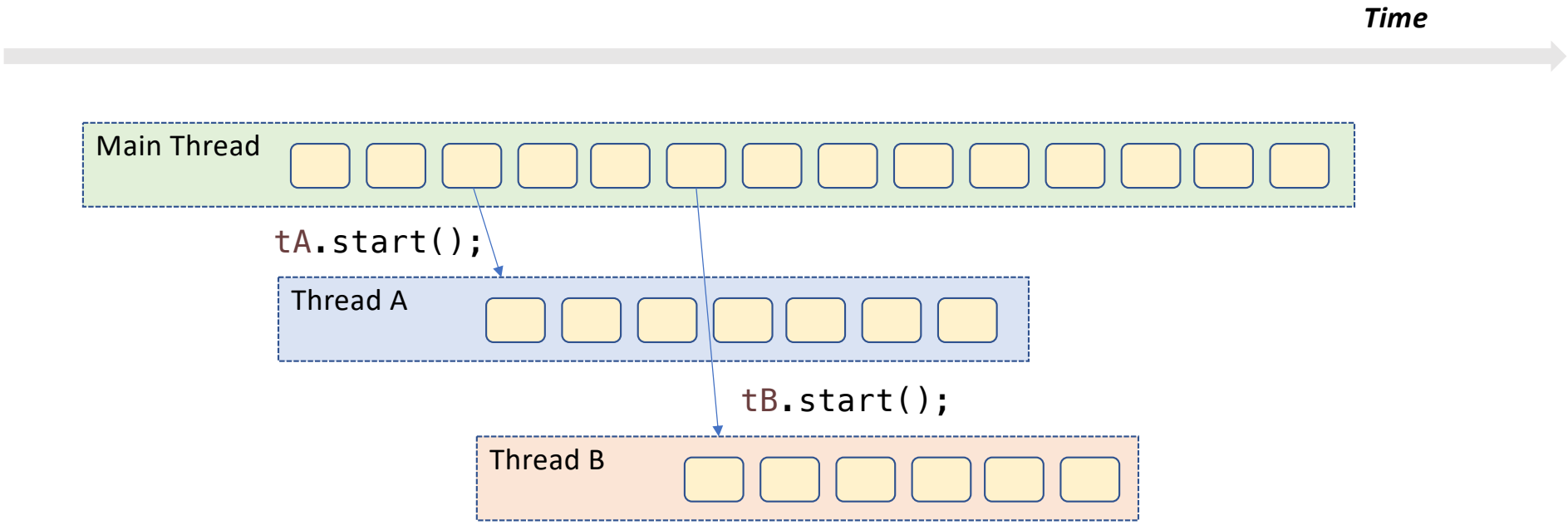
Image source: http://tutorials.jenkov.com/images/java-concurrency/java-memory-model-3.png

Why concurrency

- **Exploiting Multiple Processors**
  - Improve throughput by utilizing available processor resources more effectively.
- **Simplicity of Modeling**
  - Decompose complicated, asynchronous workflow into a number of simpler, synchronous workflows interacting only at specific synchronization points.
- **Improve GUI applications**
  - Enable more responsive user interfaces




```
public class MainThread
{
    public static void main(String[] args)
    {
        Thread t = Thread.currentThread();
        System.out.println("Current thread: " + t.getName());
    }
}
```



Each thread is associated an instance of the class Thread

Create New Thread – Method 1

To declare a class to be a subclass of Thread. This subclass should override the run method of class Thread.

```
class PrimeThread extends Thread
{
   long minPrime;
   PrimeThread(long minPrime) { this.minPrime = minPrime; }
   public void run()
   {
      // compute primes larger than minPrime  . . .
   }
}
```

Create New Thread – Method 1

Then create a thread and start it running:

```
public class MainThread
{
  public static void main(String[] args)
  {
    PrimeThread p = new PrimeThread(143);
    p.start();
  }
}
```

Create New Thread – Method 2

Which is better?

To declare a class that implements the Runnable interface. That class then implements the run method.

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

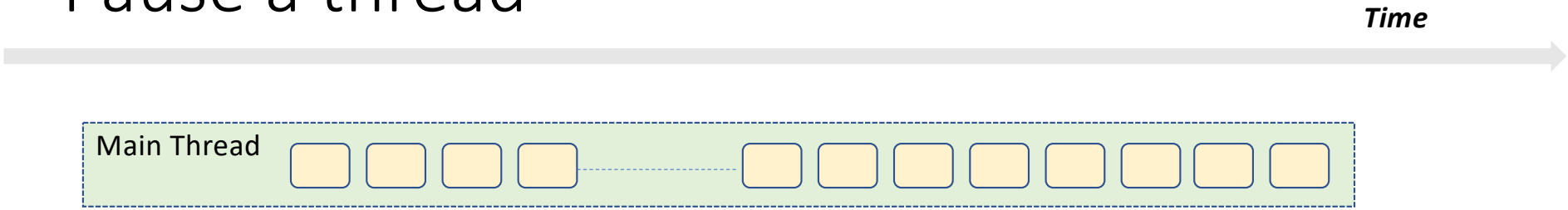
Create New Thread – Method 2

Which is better?

An instance of the class can then be allocated, passed as an argument when creating Thread, and started.

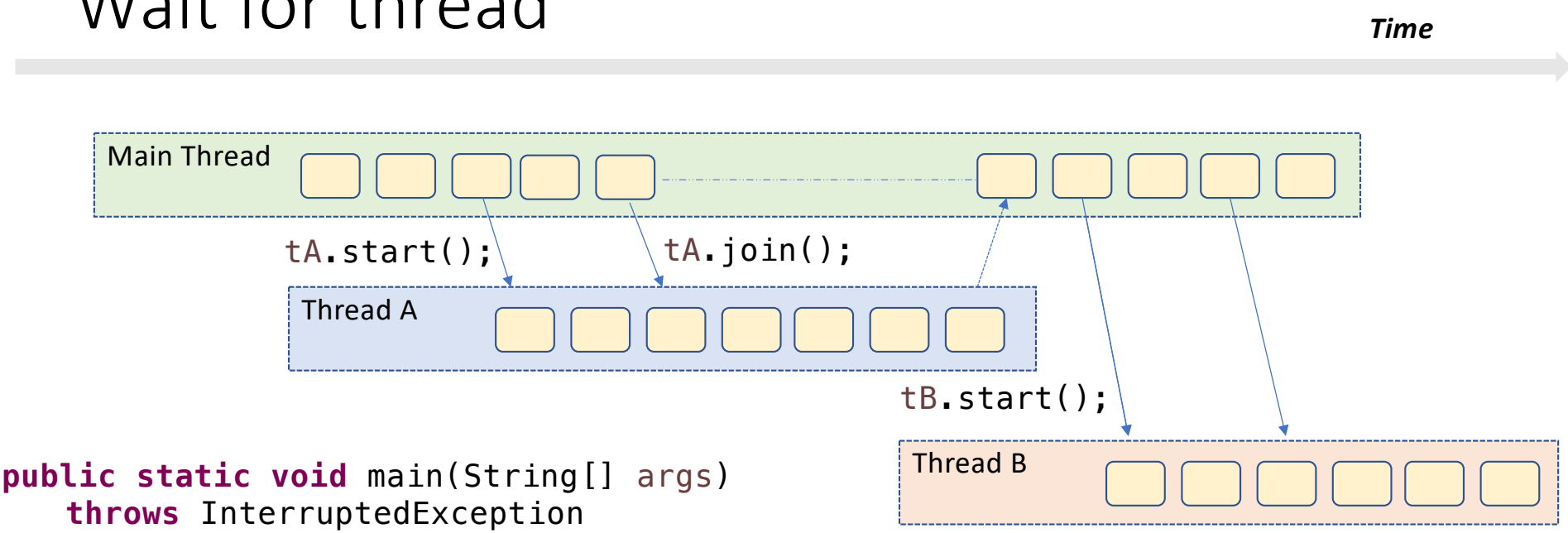
```
public class MainThread
{
   public static void main(String[] args)
   {
      PrimeRun p = new PrimeRun(143);
      new Thread(p).start();
   }
}
```

Pause a thread




```
public static void main(String[] args)
   throws InterruptedException
{
        for (int i = 0;i < 10;i++)
        {
            Thread.sleep(1000);
            System.out.println(i);
        }
}
```

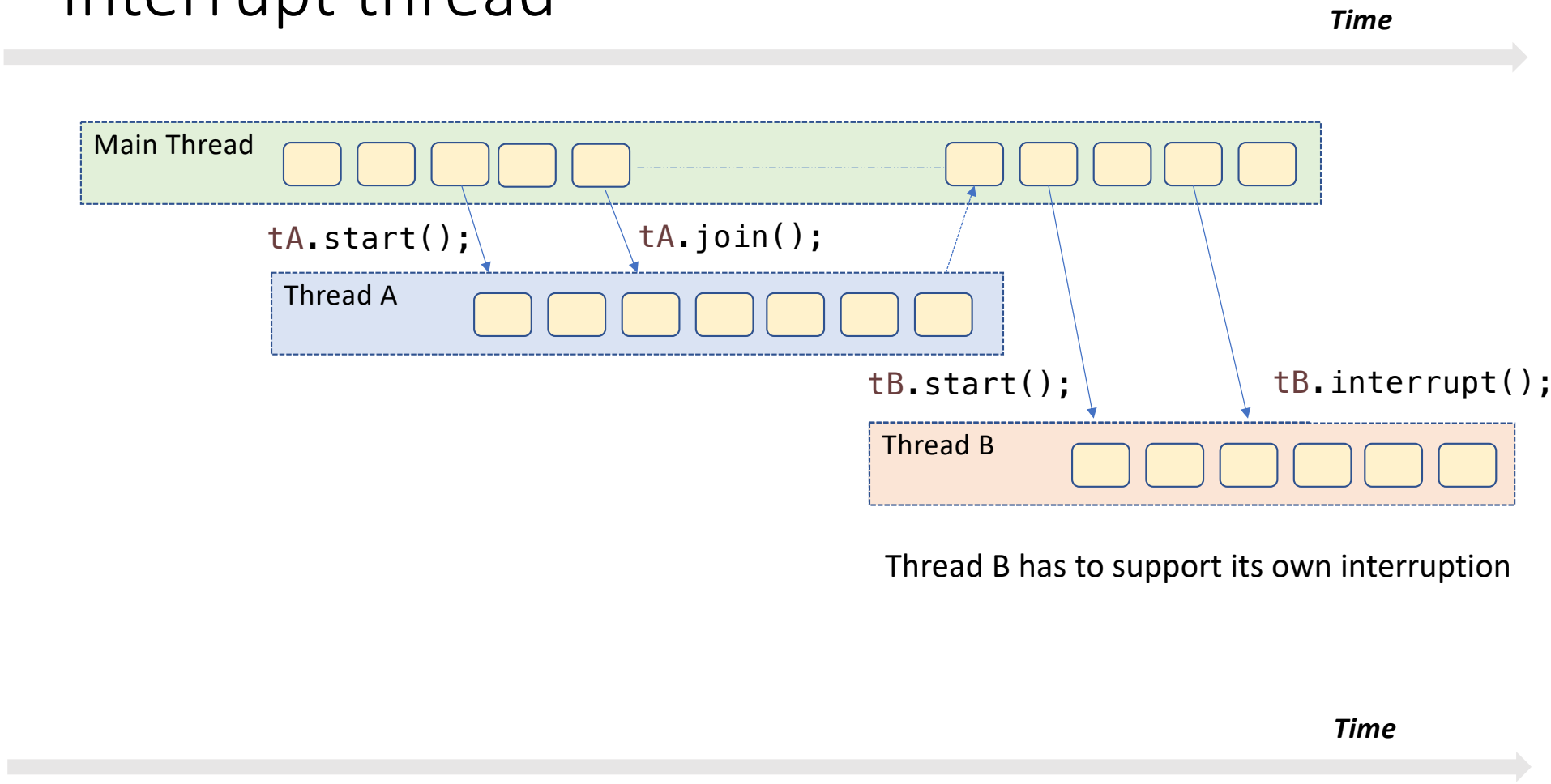
Wait for thread




```
public static void main(String[] args)
    throws InterruptedException
{
    Thread tA = new Thread(()->System.out.println("Running Thread A"));
    Thread tB = new Thread(()->System.out.println("Running Thread B"));
    tA.start();
    tA.join();
    System.out.println("Main Thread Waiting for Thread A");
    tB.start();
}
```

DistributedComputation Demo

Interrupt thread

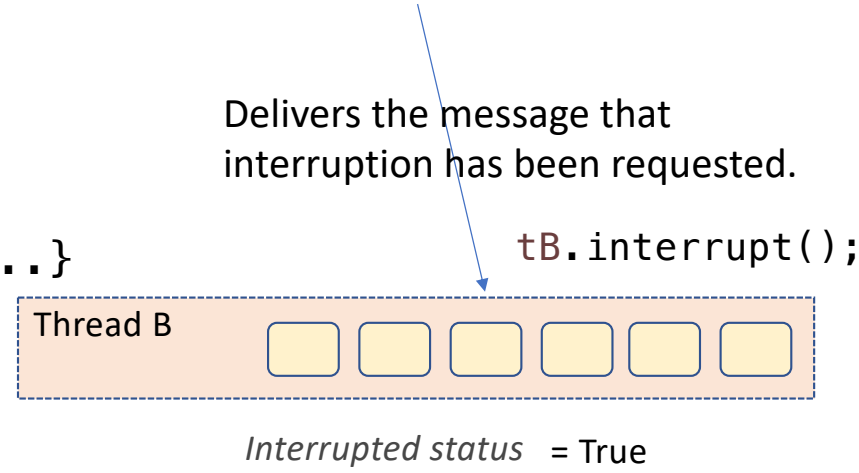


Thread B has to support its own interruption

Interrupt thread

- A cooperative mechanism

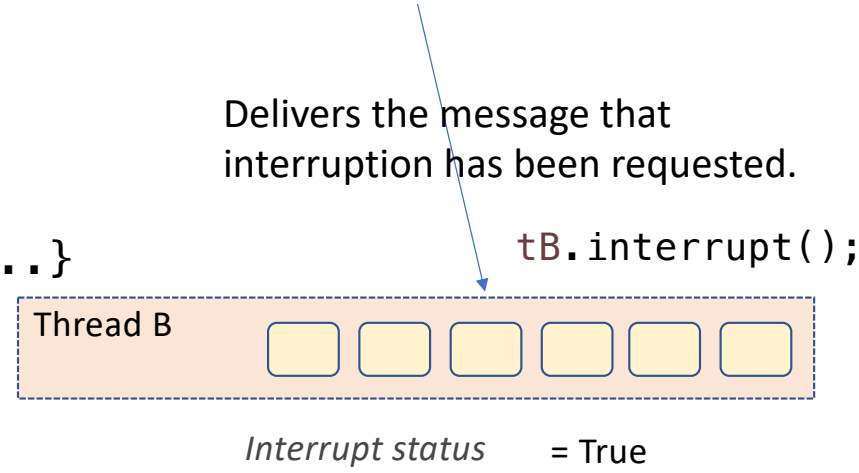
```
public class Thread {
    public void interrupt() {...}
    public boolean isInterrupted() {...}
    public static boolean interrupted() {...}
    ...
}
```



Interrupt thread

- A cooperative mechanism

```
public class Thread {
    public void interrupt() {...}
    public boolean isInterrupted() {...}
    public static boolean interrupted() {...}
    ...
}
```

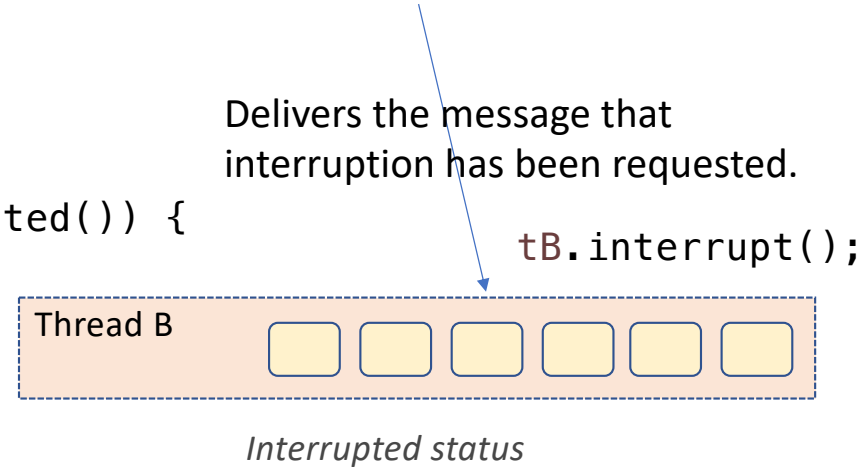


Interrupt thread

- A cooperative mechanism

```
public void run() {
    for (int i = 0; i < inputLength; i++) {
        // Heave operation
        if (Thread.currentThread().isInterrupted()) {
            return;
        }
    }
}
```

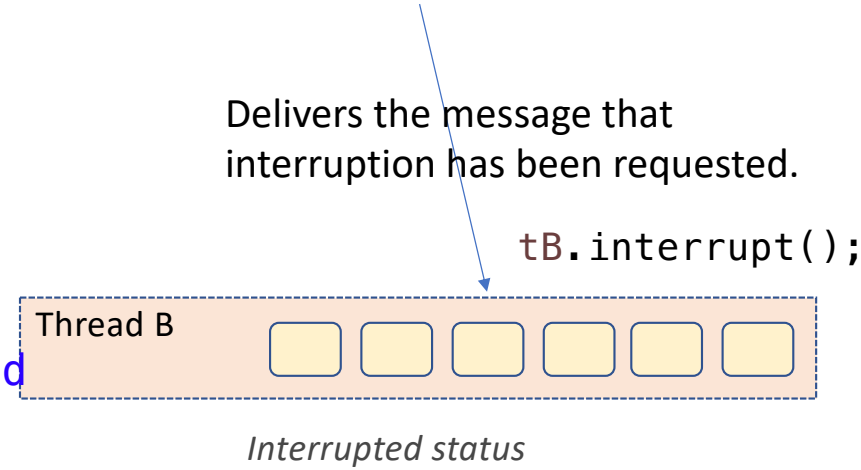
vs `if (Thread.Interrupted())`



Interrupt thread

- A cooperative mechanism

```
public void run()
{
   try
   {
      Thread.sleep(5000);
      System.out.println("Thread completed
      normally");
   }
   catch(InterruptedException e)
   {
      System.out.println("Thread interrupted");
   }
}
```



InterruptedThread and JoiningHands Demo