Thread I

Slides courtesy of Dr. Nilanjan Banerjee

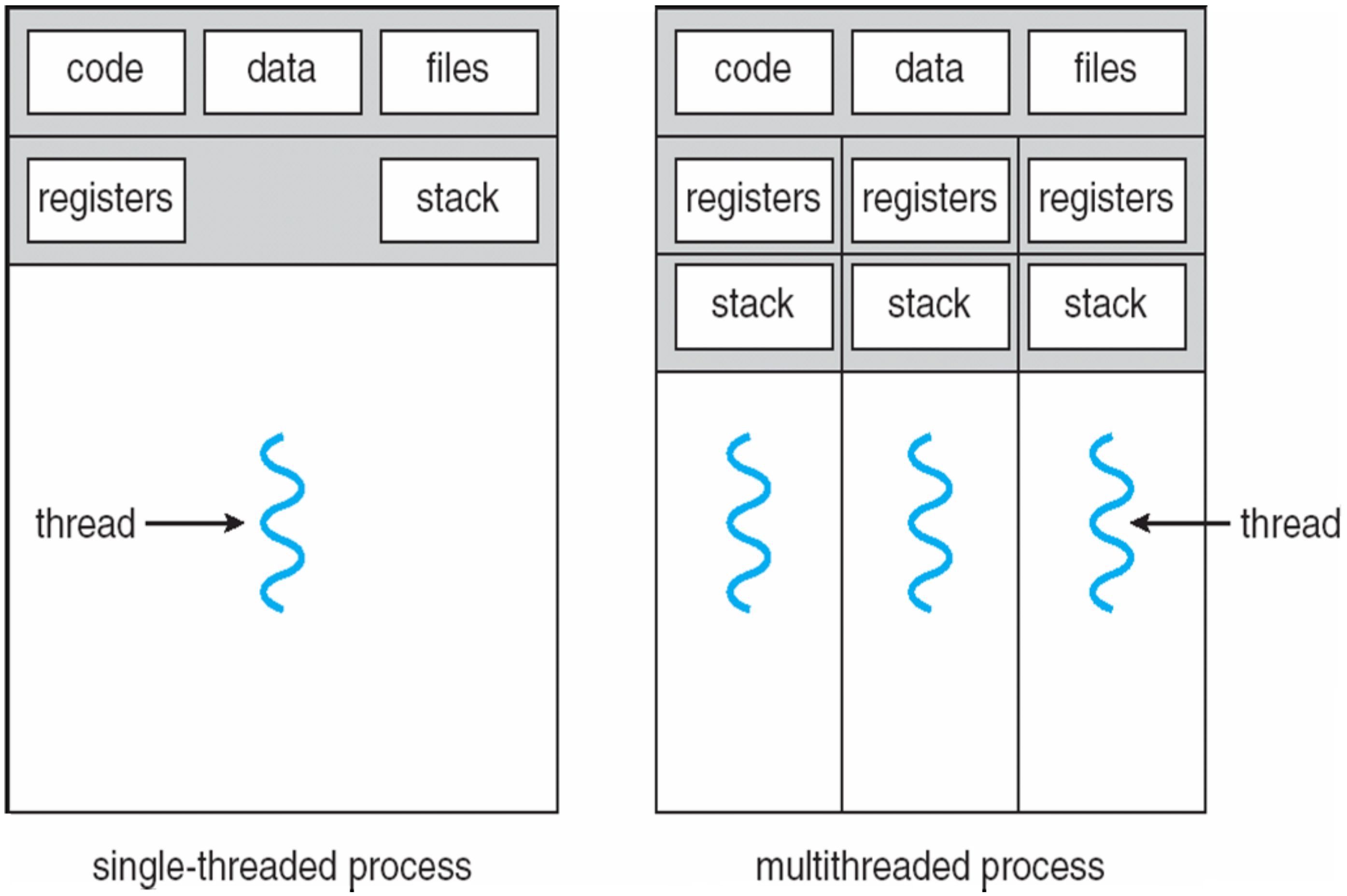
Concurrent Programming

- Running tasks in parallel
  - Multiple processes
  - Multiple threads in one process?

- Advantage: speed if you can decompose a task into parallel independent tasks

- Examples
  - Webserver: multiple requests to cater to.
  - Web browser: multiple objects loading simultaneously
  - Assignment 4: quickly cracking a hash of a password?
  - Encoding multiple blocks simultaneously
  - Almost everything today can be broken into parallel tasks.

Threads and running processes



Creating Threads: **extend** the Thread class

- Thread is a basic class which provides two methods

  void start()

  - Creates a new thread and makes it runnable

  void run()

  - A new Thread begins its life inside this method

```
public class A extends Thread {
    public A() { … constructor…}
    public void run() {…method that will run in the thread….}

}
```

Lets take a concrete example in eclipse

Application: Provide parallel addition

Input: array of integers a[1,…,k]

Divide array into segments and in parallel add segments of the array

For example:

Thread 1: add a[1:k/2]

Theads 2: add a[k/2:k]

Add both of them together

Thread Creation Diagram

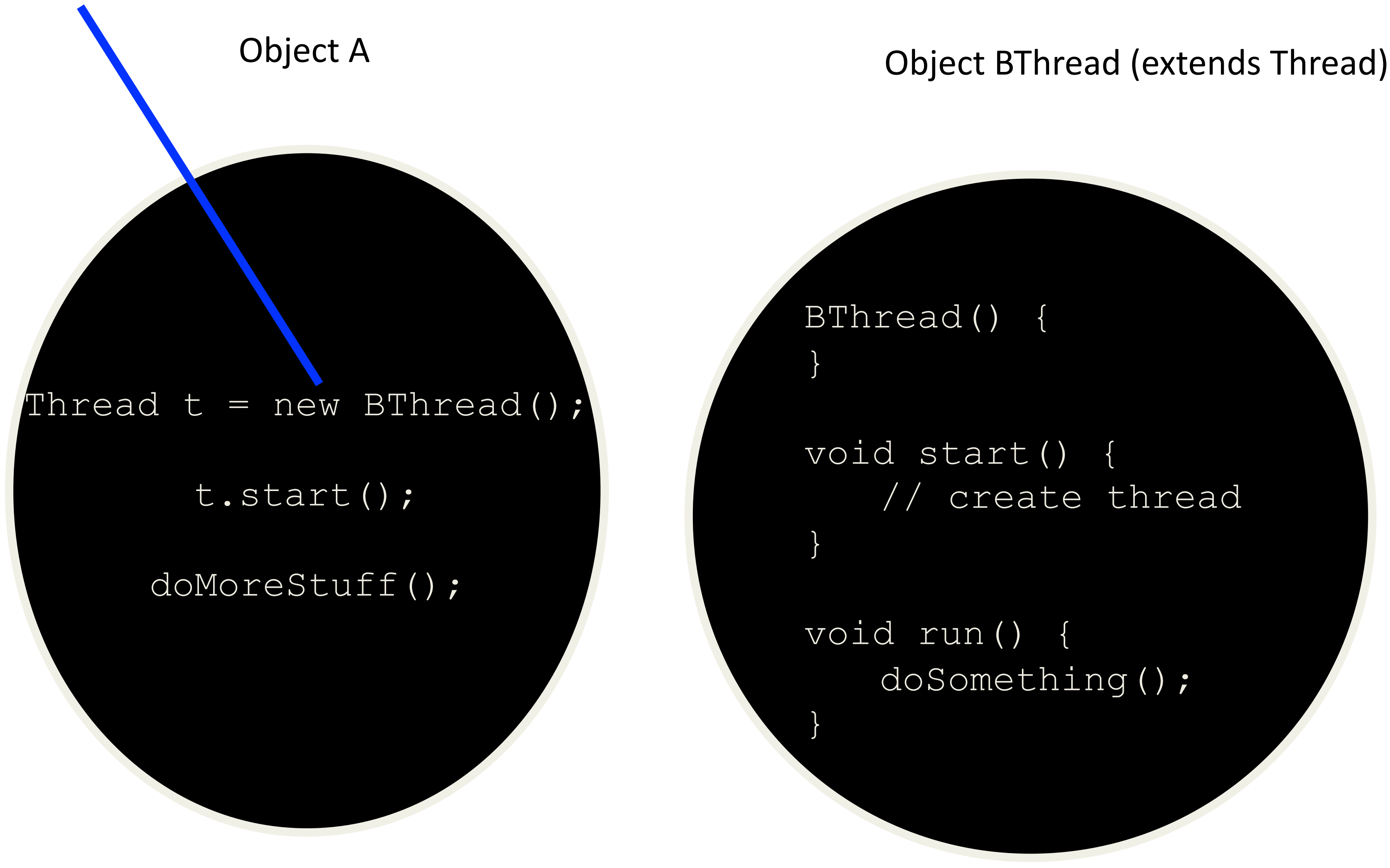
Object A

```
Thread t = new BThread();

t.start();

doMoreStuff();
```

Object BThread (extends Thread)

```
BThread() {
}

void start() {
   // create thread
}

void run() {
   doSomething();
}
```

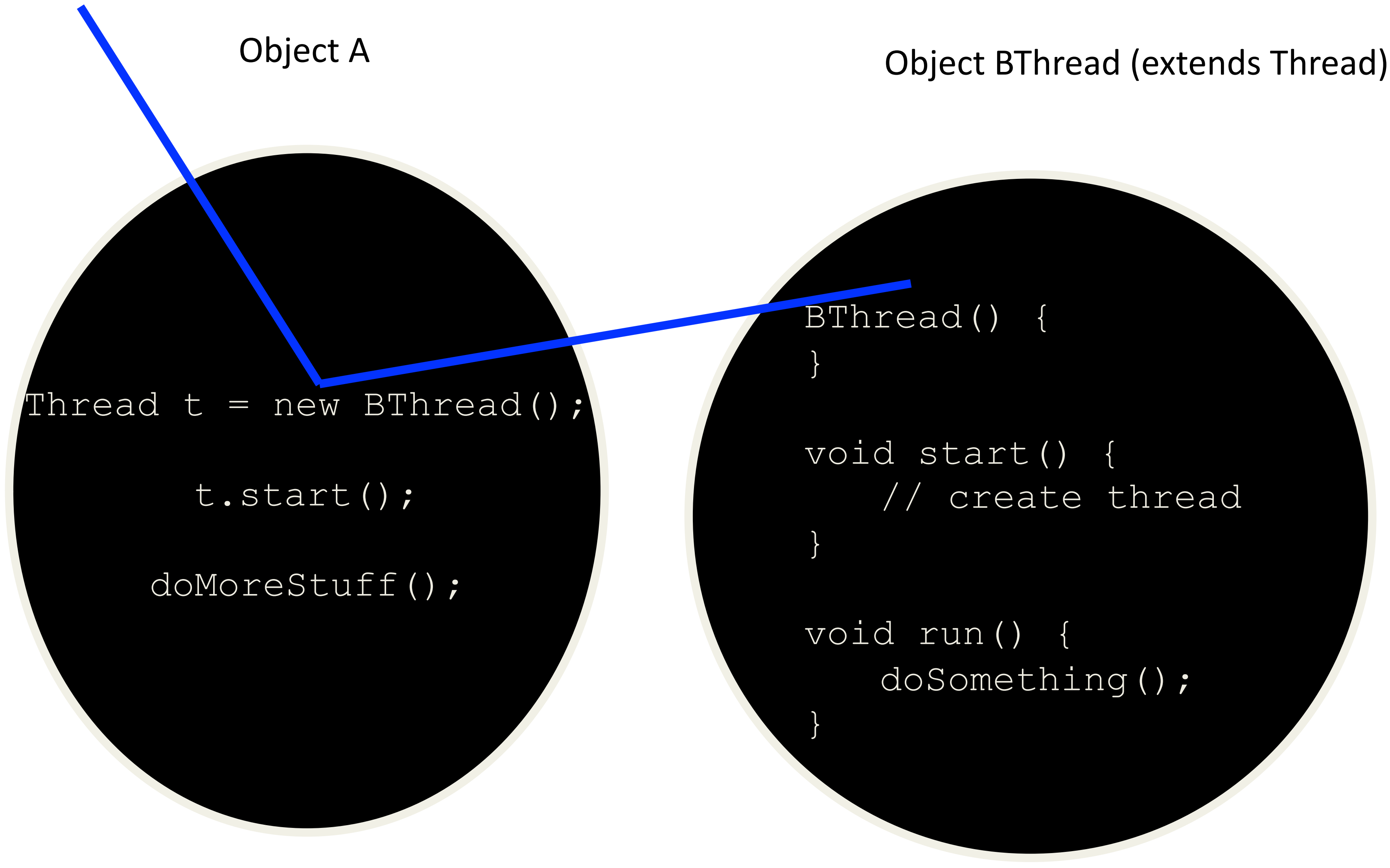
Thread Creation Diagram



```
Thread t = new BThread();

t.start();

doMoreStuff();
```

```
BThread() {
}

void start() {
    // create thread
}

void run() {
    doSomething();
}
```

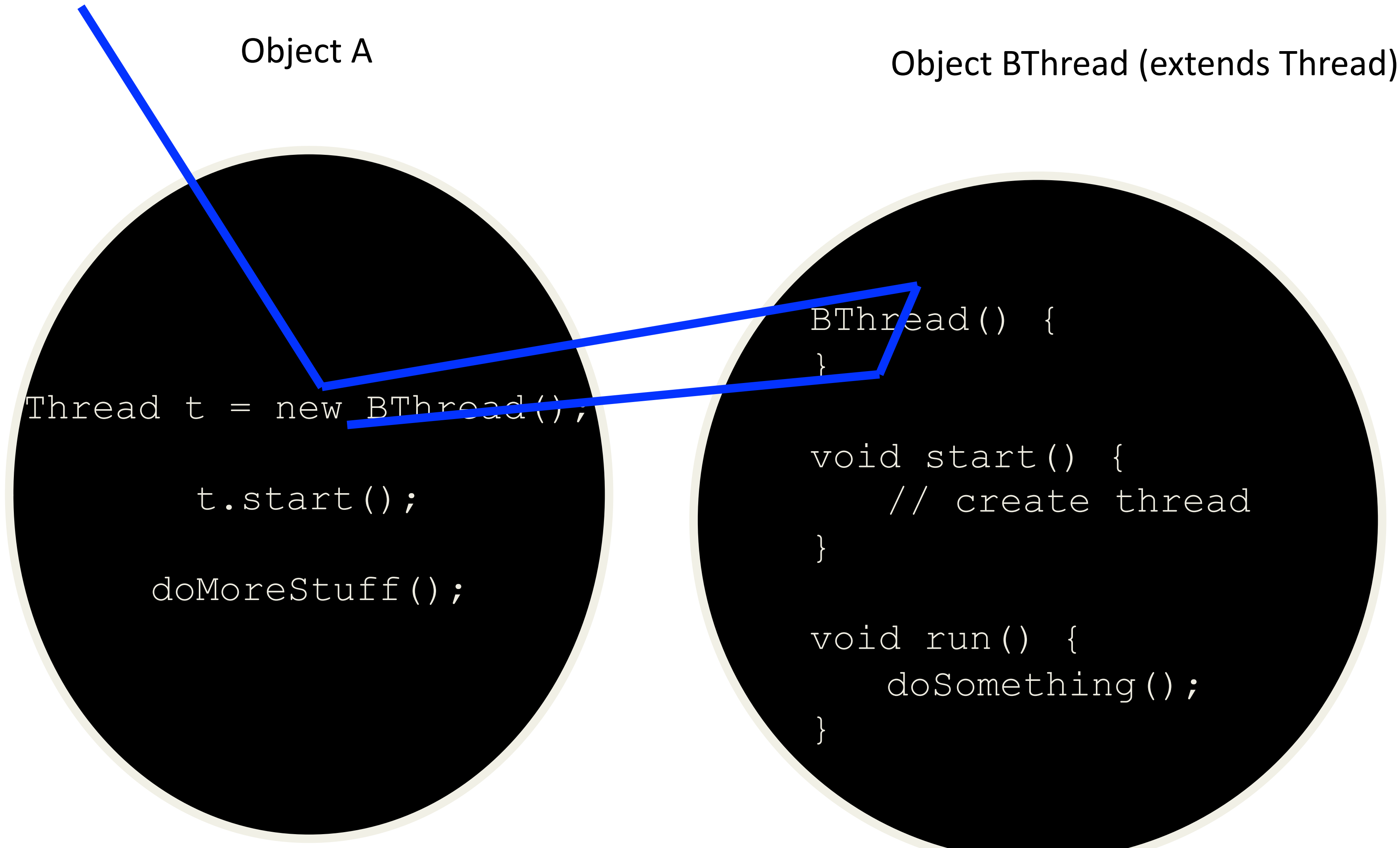
Thread Creation Diagram



```
Thread t = new BThread();

t.start();

doMoreStuff();
```

```
BThread() {
}

void start() {
    // create thread
}

void run() {
    doSomething();
}
```

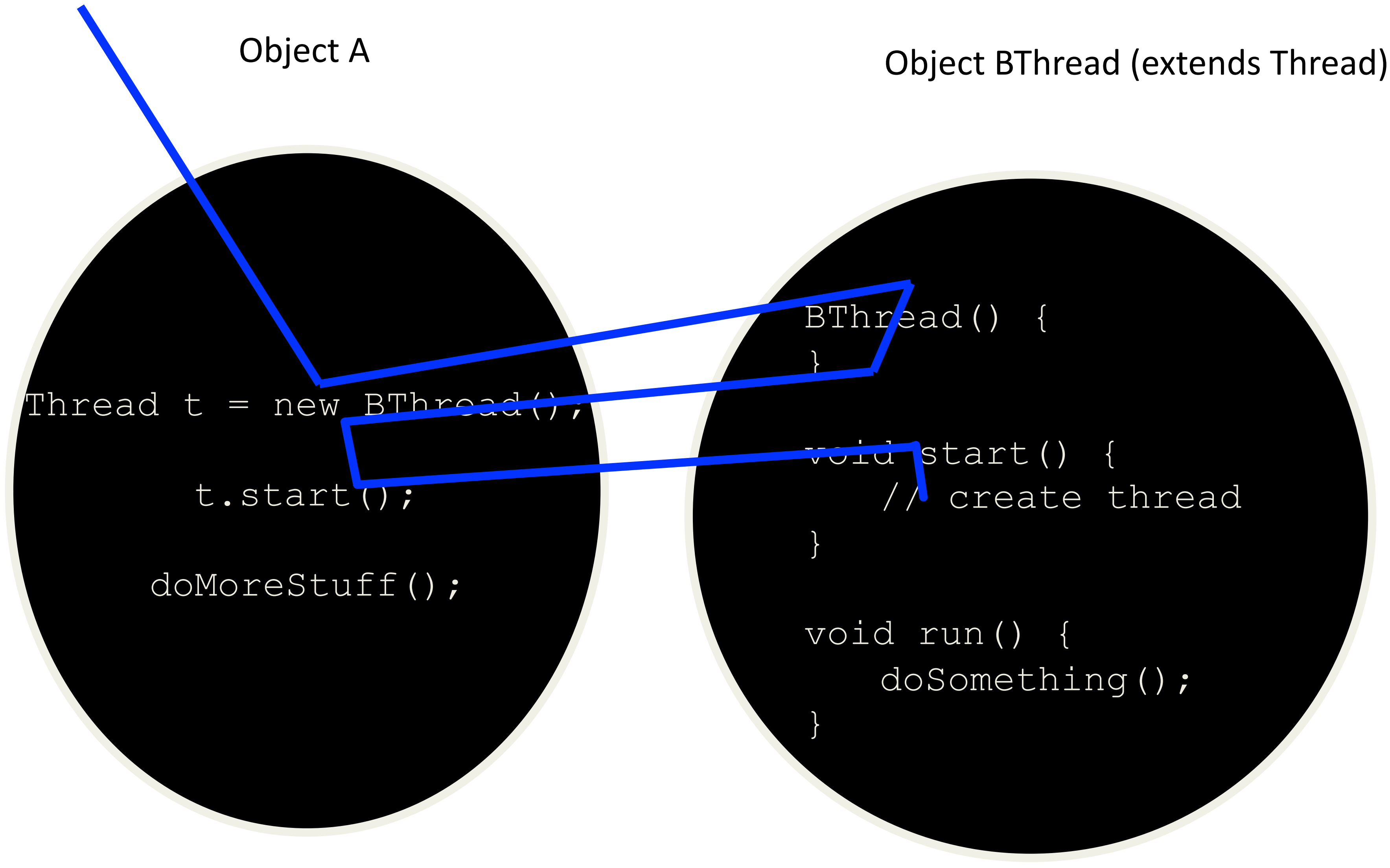
Thread Creation Diagram



```
Thread t = new BThread();

t.start();

doMoreStuff();
```

```
BThread() {
}

void start() {
    // create thread
}

void run() {
    doSomething();
}
```

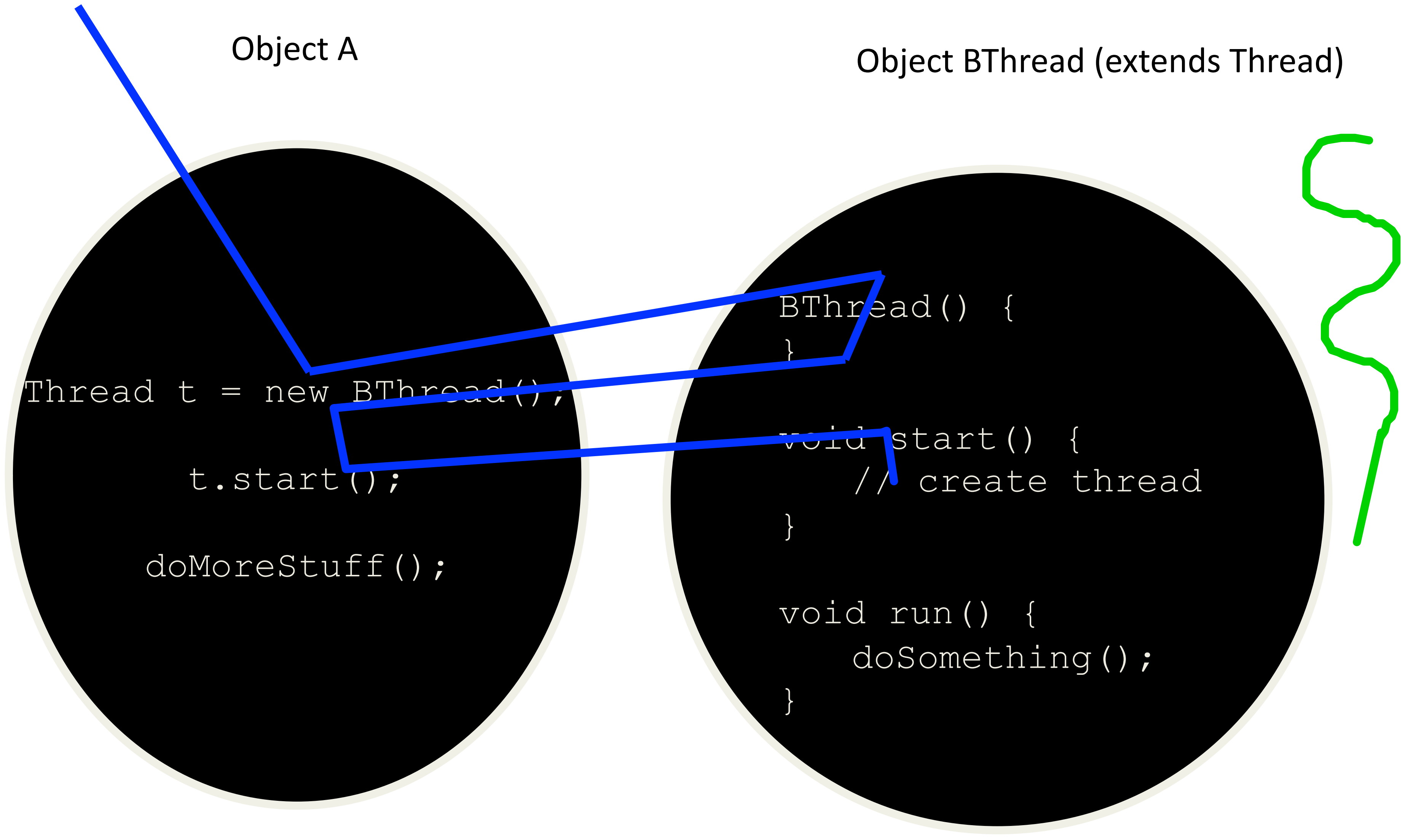
Thread Creation Diagram



```
Thread t = new BThread();

    t.start();

    doMoreStuff();
```

```
BThread() {
}

void start() {
    // create thread
}

void run() {
    doSomething();
}
```

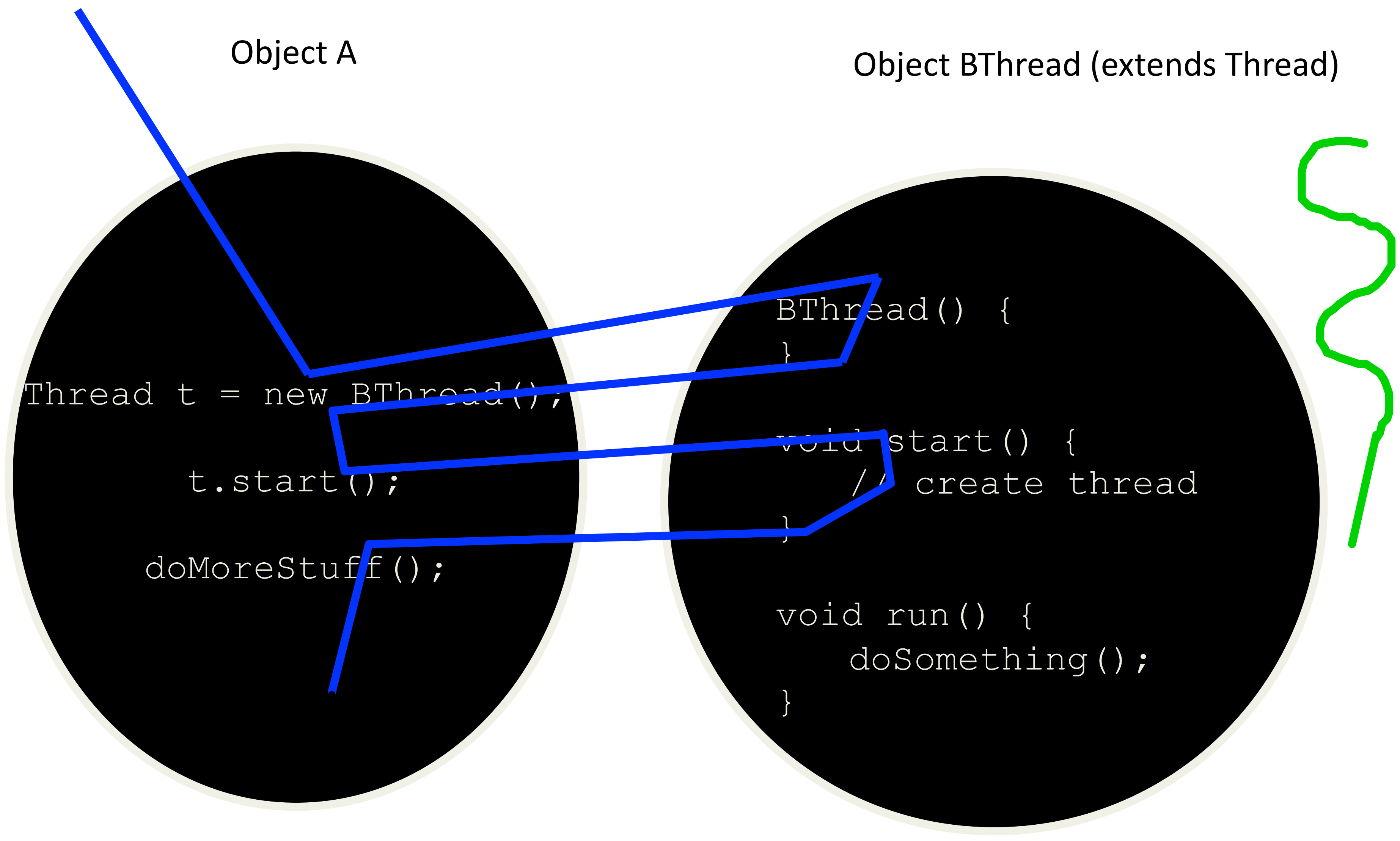
Thread Creation Diagram



```
Thread t = new BThread();

      t.start();

   doMoreStuff();
```

```
BThread() {
}

void start() {
    // create thread
}

void run() {
    doSomething();
}
```

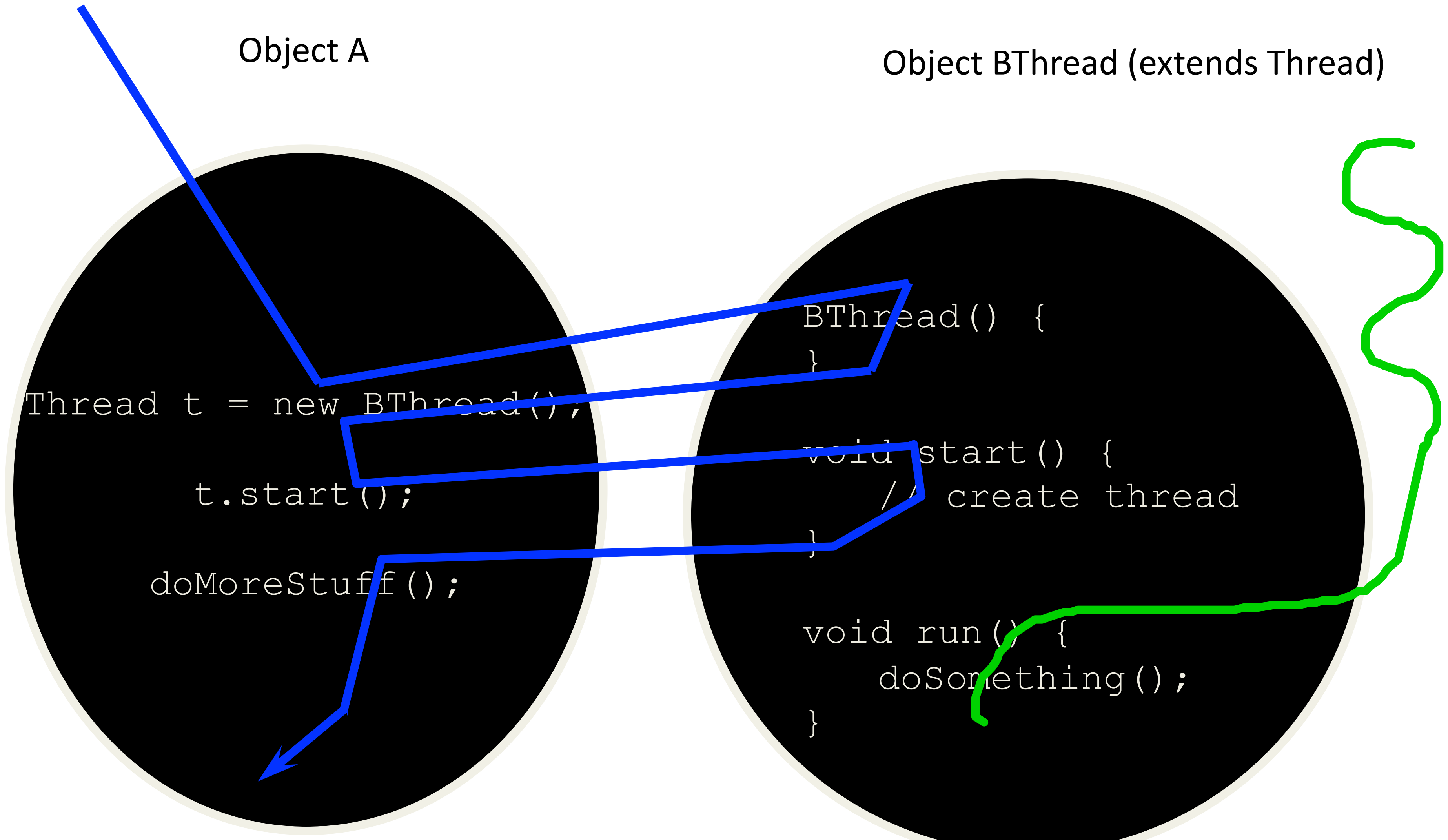
Thread Creation Diagram



```
Thread t = new BThread();

      t.start();

    doMoreStuff();
```

```
BThread() {
}

void start() {
    // create thread
}

void run() {
    doSomething();
}
```

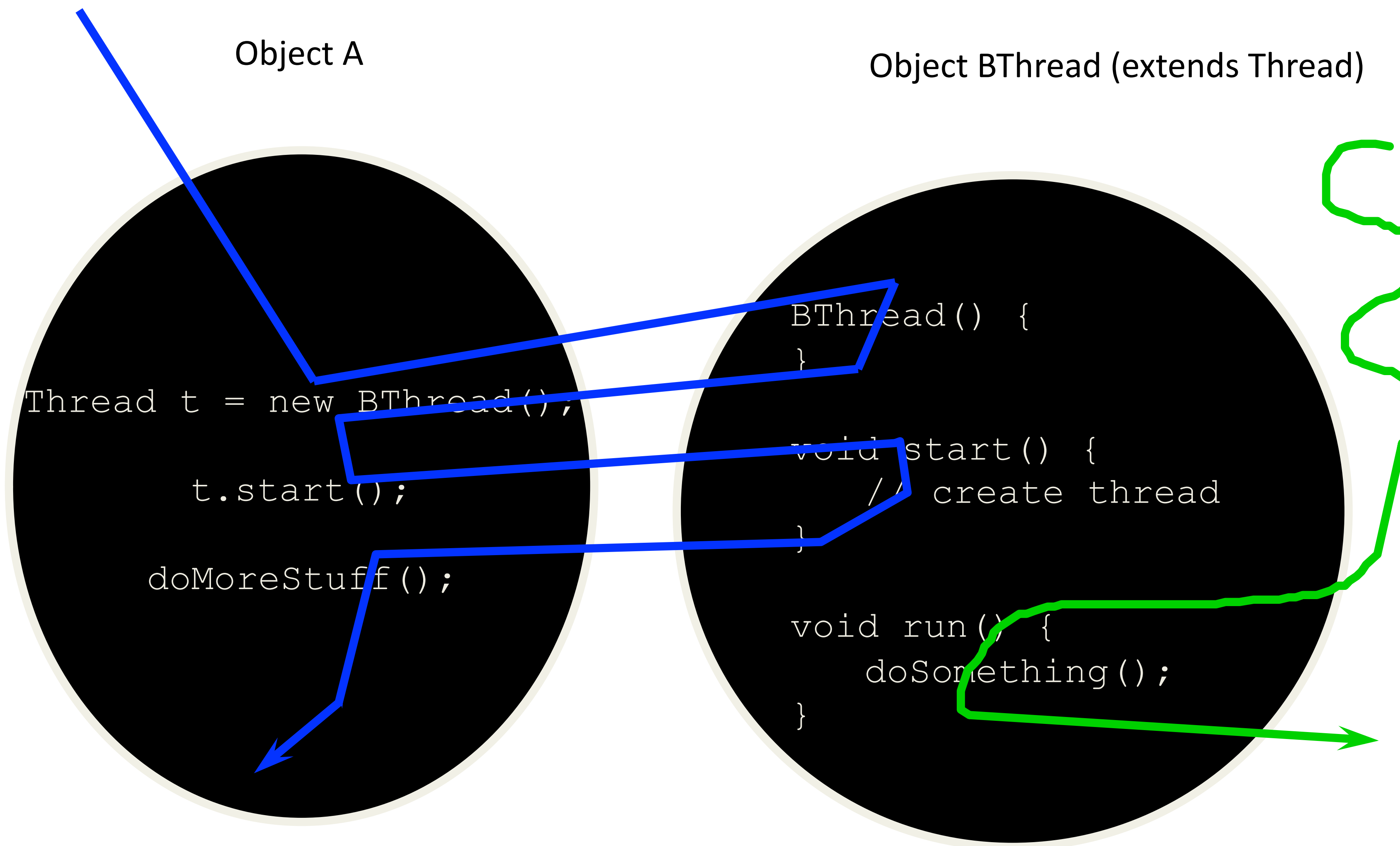
Thread Creation Diagram



```
Thread t = new BThread();

      t.start();

    doMoreStuff();
```

```
BThread() {
}

void start() {
    // create thread
}

void run() {
    doSomething();
}
```

Thread Creation Diagram



```
Thread t = new BThread();

    t.start();

  doMoreStuff();
```

```
BThread() {
}

void start() {
    // create thread
}

void run() {
    doSomething();
}
```

Another way of creating Threads: **Runnable** interface

- A helper to the thread object
- Your class implements the Runnable interface
- The Thread object’s run() method calls the Runnable object’s run() method
- Allows threads to run inside any object, regardless of inheritance

Example of using the **Runnable** interface

```
public class MyRunnable implements Runnable {
  String name;
  public MyRunnable(String name) {
    this.name = name;
  }
  public void run() {
    for(int i; i < 10; i++) {
      System.out.println(i + “ “ + name());
      try {
        sleep((long)(Math.random() * 1000));
      } catch(InterruptedException e) {}
    }
  }
}

public class ThreadTest {
  public static void main(String[] args) {
    for(int i = 0; i < args.length; i++) {
      Thread t = new Thread(new MyRunnable(args[i]), args[i]);
      t.start();
    }
  }
}
```

Blocking Threads

- When reading from a stream, if input is not available, the thread will <u>block</u>
- Thread is suspended ("blocked") until I/O is available
- Allows other threads to automatically activate
- When I/O available, thread wakes back up again
  - Becomes "<u>runnable</u>"
  - Not to be confused with the Runnable interface

Thread Scheduling

- In general, the <u>runnable</u> thread with the highest <u>priority</u> is active (running)
- Java is <u>priority-preemptive</u>
  - If a high-priority thread wakes up, and a low-priority thread is running
  - Then the high-priority thread gets to run immediately
- Allows on-demand processing
  - Efficient use of CPU

Thread Starvation

- If a high priority thread never blocks
- Then all other threads will starve
- Must be clever about thread priority

Thread Priorities: General Strategies

- Threads that have more to do should get lower priority
- Counterintuitive
- Cut to head of line for short tasks
- Give your I/O-bound threads high priority
  - Wake up, immediately process data, go back to waiting for I/O

Thread interruption

- Threads execution exits when the run() method returns
- Or if it throws an exception that is not handled in the run() method
- What if you want to interrupt a running Thread?
- Thread.interrupt() --- call interrupts a Thread
  - Sets a interrupt flag for the Thread object!
  - How does a Thread check whether the flag is checked?
  - Thread.currentThread.isInterrupted()?

Example

```
while (!Thread.currentThread().isInterrupted())
{ …do something ….}
```

What if the Thread is sleeping or blocked?
Solution: ***catch InterruptedException***?

```
try { while(!Thread.currentThread.isInterrupted()) { ..do something…}
catch(InterruptedException e) { //thread interrupted during sleep or wait}
```

When interrupted, interrupt flag is set and the Thread is woken up!

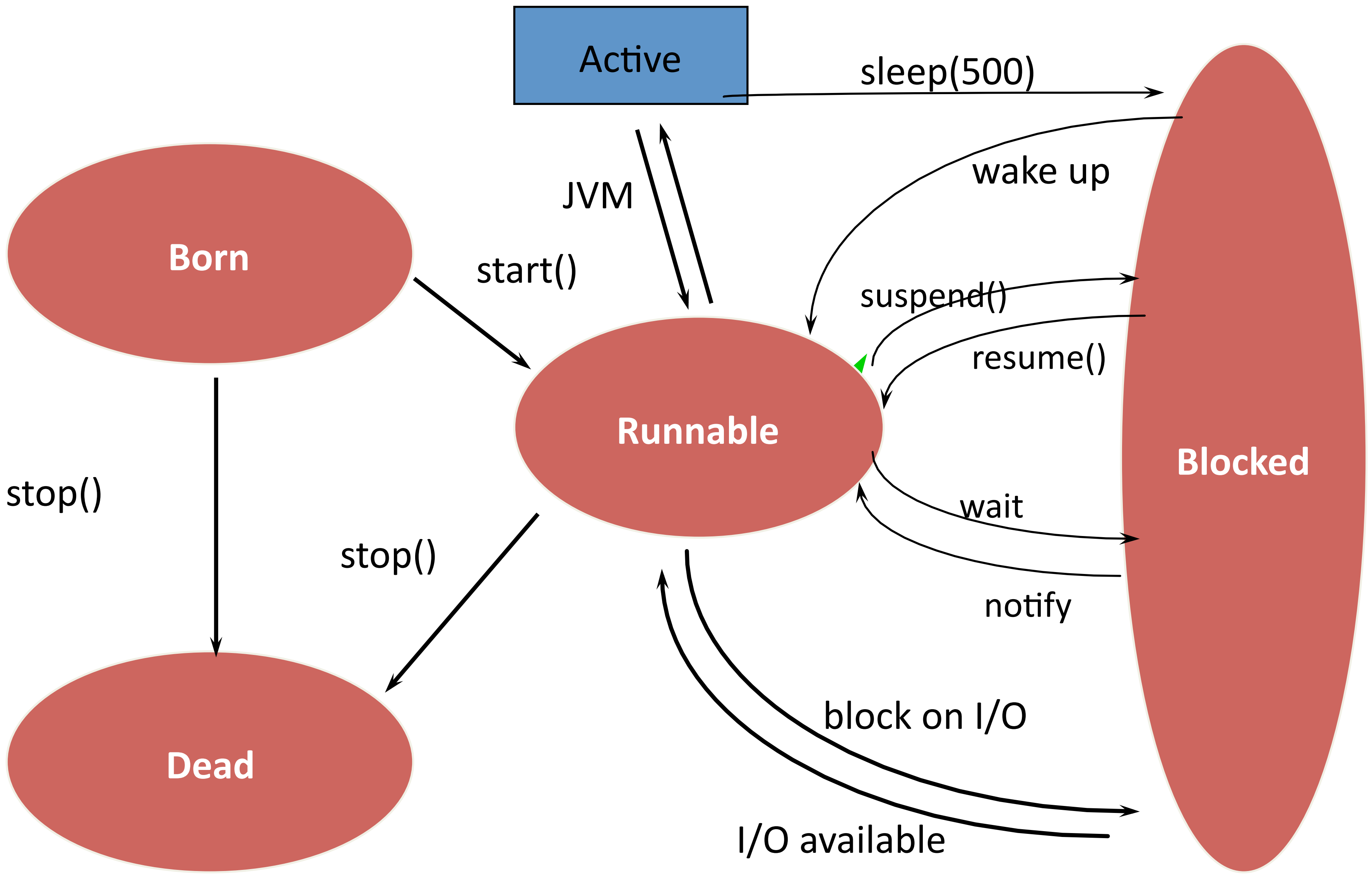
Race Conditions

- Two threads are simultaneously modifying a single object
- Both threads “race” to store their value
- In the end, the last one there “wins the race”
- (Actually, both lose)

Race Condition Example

Lets take an example in eclipse to illustrate this

Thread Lifecycle


Active
sleep(500)
JVM
wake up
Born
start()
suspend()
resume()
Runnable
Blocked
stop()
wait
stop()
notify
block on I/O
Dead
I/O available