

# Shared Memory Programming

## Java Threads

### Lec 1

Mohsan Jameel, Information Systems and  
Machine Learning Lab, University of  
Hildesheim

1

## What is a Process?

### ❑ A *process* is an “instance” of a program running.

- ❖ Modern OSes run multiple processes simultaneously

### ❑ Examples (can all run simultaneously):

- ❖ gcc file\_A.c – compiler running on file A
- ❖ gcc file\_B.c – compiler running on file B
- ❖ emacs – text editor
- ❖ firefox – web browser

### ❑ Non-examples (implemented as one process):

- ❖ Multiple firefox tabs are part of one process.

### ❑ Why processes?

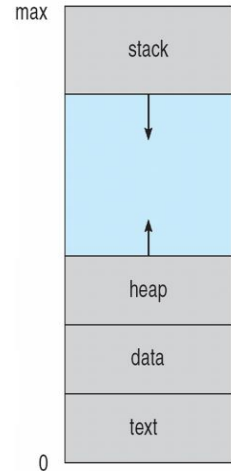
- ❖ Simplicity of programming
- ❖ Higher throughput (better CPU utilization), lower latency

Mohsan Jameel, Information Systems and  
Machine Learning Lab, University of  
Hildesheim

2

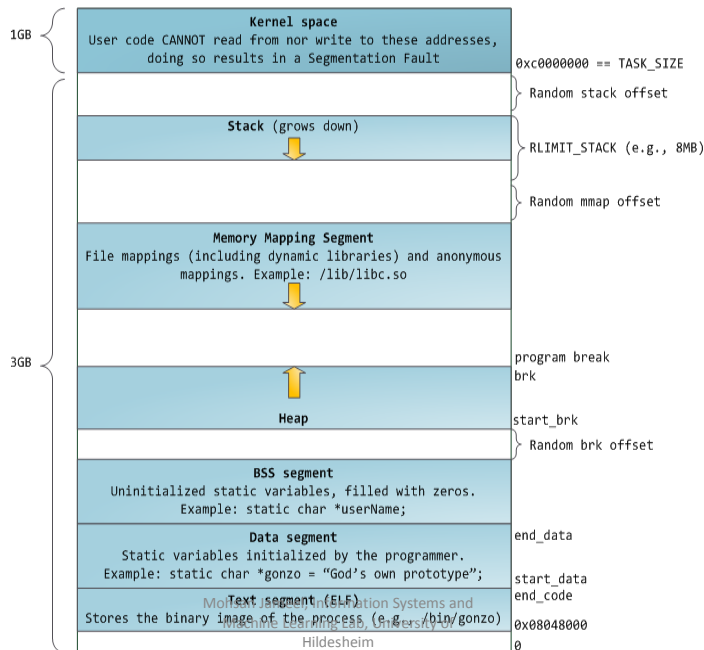
# What is a Process?

- ❑ **Each proc.  $P_i$  has own view of machine**
  - ❖ Its own address space.
  - ❖ Its own open files.
  - ❖ Its own virtual CPU (through preemptive multitasking)
  - ❖ `*(char *)0xc000` **different in  $P_1$  &  $P_2$**
- ❑ **Greatly simplifies programming model**
  - ❖ gcc does not care that firefox is running
- ❑ **Sometimes want interaction between processes**
  - ❖ Simplest is through files: emacs edits file, gcc compiles it
  - ❖ More complicated: Shell/command, Window manager/app.



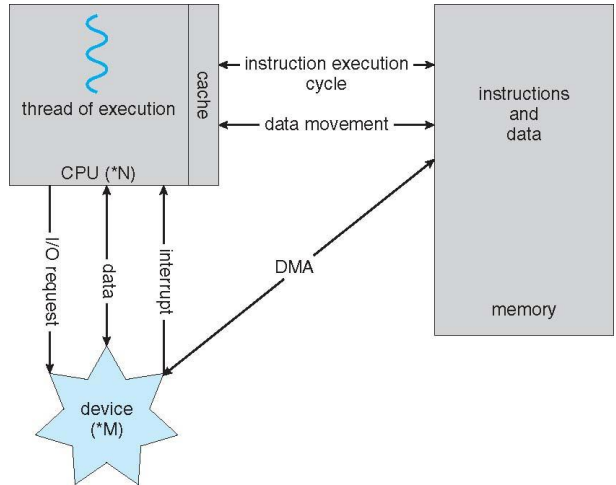
Mohsan Jameel, Information Systems and Machine Learning Lab, University of Hildesheim

# Process Organization in Memory



Mohsan Jameel, Information Systems and Machine Learning Lab, University of Hildesheim

# Basic Execution



Mohsan Jameel, Information Systems and Machine Learning Lab, University of Hildesheim

# Basic Execution Environment

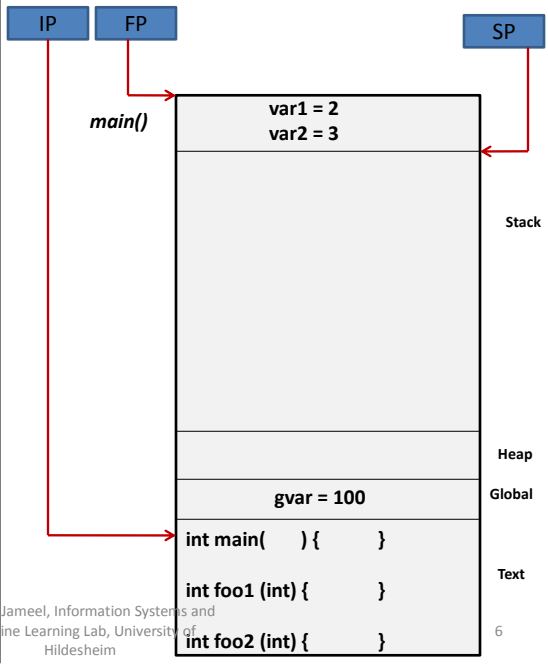
```

int gvar = 100;

int foo2 (int b)
{
    return b * gvar;
}

int foo1 (int a)
{
    int lvar = a + gvar;
    return foo2(lvar);
}

int main ( )
{
    int var1, int var2;
    var1 = 2;
    var2 = 3;
    var1 = foo1(var1);
    var2 = foo1(var2);
    return 0;
}
    
```



Mohsan Jameel, Information Systems and Machine Learning Lab, University of Hildesheim

## Basic Execution Environment

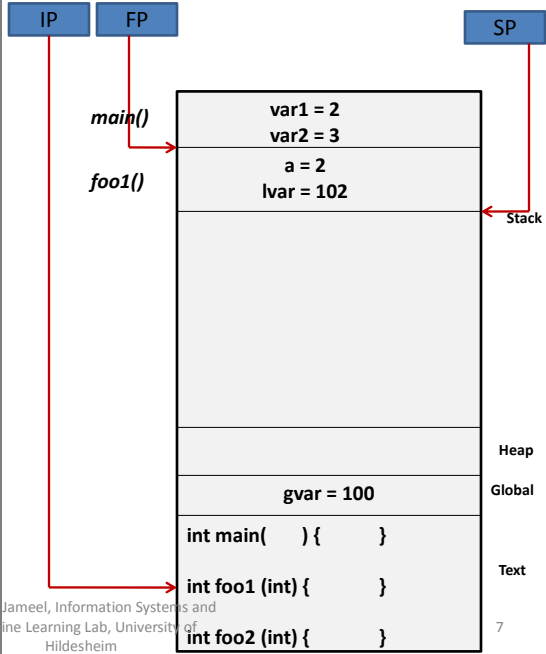
```

int gvar = 100;

int foo2 (int b)
{
    return b * gvar;
}

int foo1 (int a)
{
    int lvar = a + gvar;
    return foo2(lvar);
}

int main ( )
{
    int var1, int var2;
    var1 = 2;
    var2 = 3;
    var1 = foo1(var1);
    var2 = foo1(var2);
    return 0;
}
    
```



Mohsan Jameel, Information Systems and Machine Learning Lab, University of Hildesheim

7

## Basic Execution Environment

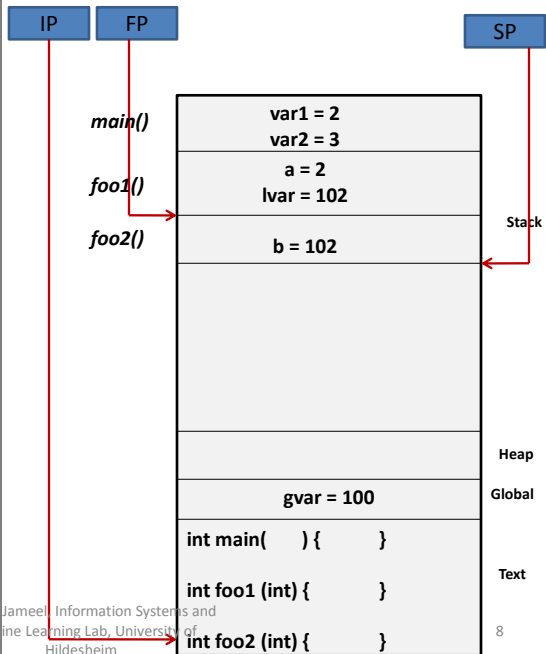
```

int gvar = 100;

int foo2 (int b)
{
    return b * gvar;
}

int foo1 (int a)
{
    int lvar = a + gvar;
    return foo2(lvar);
}

int main ( )
{
    int var1, int var2;
    var1 = 2;
    var2 = 3;
    var1 = foo1(var1);
    var2 = foo1(var2);
    return 0;
}
    
```



Mohsan Jameel, Information Systems and Machine Learning Lab, University of Hildesheim

8

## Basic Execution Environment

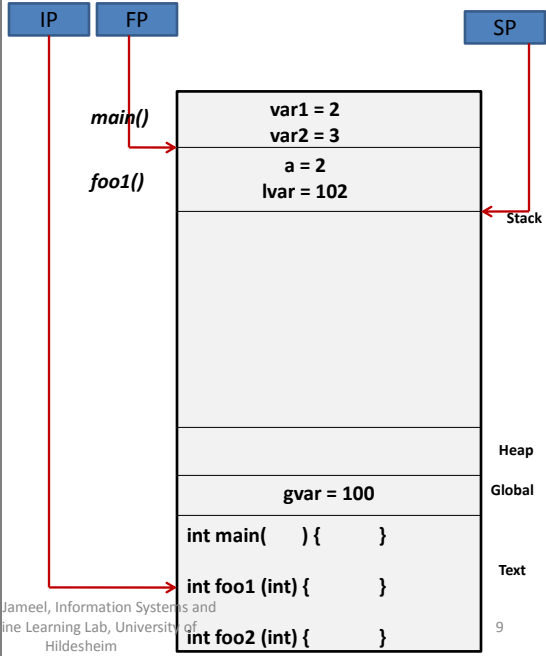
```

int gvar = 100;

int foo2 (int b)
{
    return b * gvar;
}

int foo1 (int a)
{
    int lvar = a + gvar;
    return foo2(lvar);
}

int main ( )
{
    int var1, int var2;
    var1 = 2;
    var2 = 3;
    var1 = foo1(var1);
    var2 = foo1(var2);
    return 0;
}
    
```



Mohsan Jameel, Information Systems and Machine Learning Lab, University of Hildesheim

## Basic Execution Environment

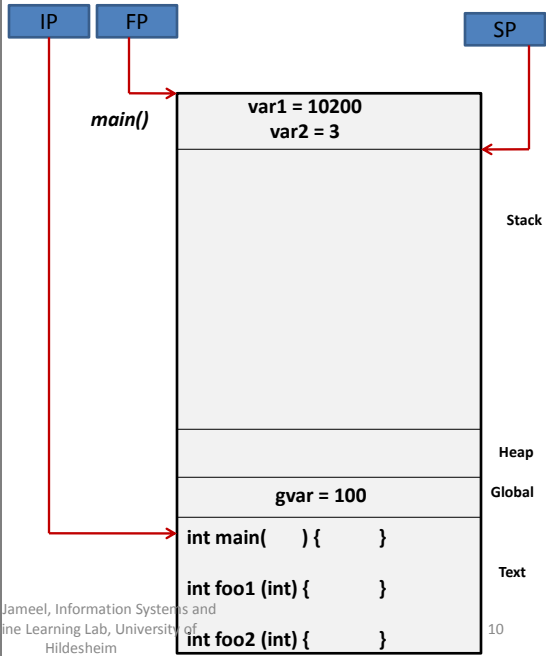
```

int gvar = 100;

int foo2 (int b)
{
    return b * gvar;
}

int foo1 (int a)
{
    int lvar = a + gvar;
    return foo2(lvar);
}

int main ( )
{
    int var1, int var2;
    var1 = 2;
    var2 = 3;
    var1 = foo1(var1);
    var2 = foo1(var2);
    return 0;
}
    
```



Mohsan Jameel, Information Systems and Machine Learning Lab, University of Hildesheim

## Basic Execution Environment

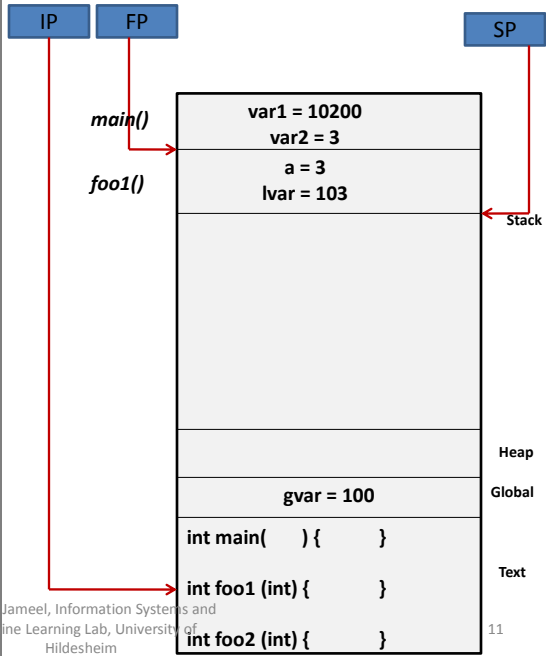
```

int gvar = 100;

int foo2 (int b)
{
    return b * gvar;
}

int foo1 (int a)
{
    int lvar = a + gvar;
    return foo2(lvar);
}

int main ( )
{
    int var1, int var2;
    var1 = 2;
    var2 = 3;
    var1 = foo1(var1);
    var2 = foo1(var2);
    return 0;
}
    
```



Mohsan Jameel, Information Systems and Machine Learning Lab, University of Hildesheim

11

## Basic Execution Environment

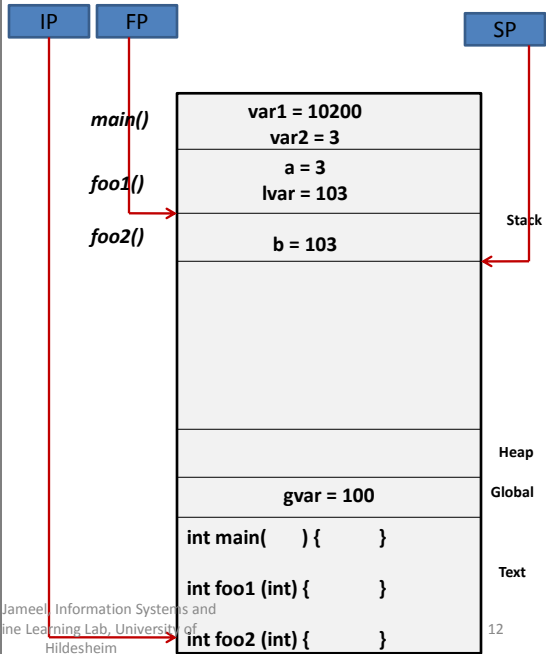
```

int gvar = 100;

int foo2 (int b)
{
    return b * gvar;
}

int foo1 (int a)
{
    int lvar = a + gvar;
    return foo2(lvar);
}

int main ( )
{
    int var1, int var2;
    var1 = 2;
    var2 = 3;
    var1 = foo1(var1);
    var2 = foo1(var2);
    return 0;
}
    
```



Mohsan Jameel, Information Systems and Machine Learning Lab, University of Hildesheim

12

## Basic Execution Environment

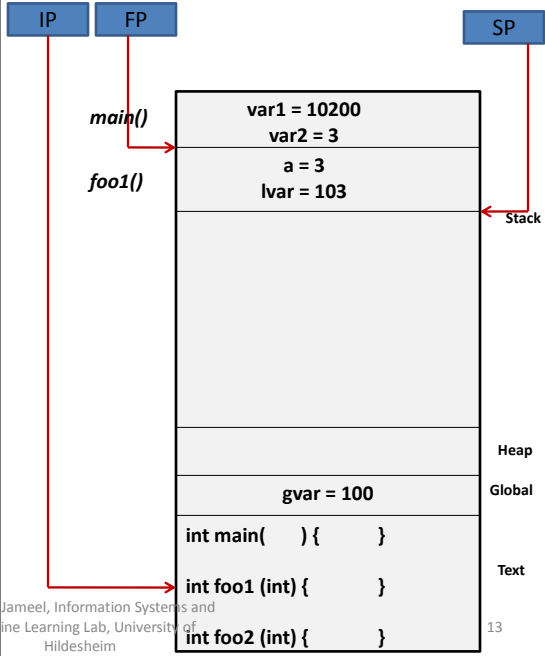
```

int gvar = 100;

int foo2 (int b)
{
    return b * gvar;
}

int foo1 (int a)
{
    int lvar = a + gvar;
    return foo2(lvar);
}

int main ( )
{
    int var1, int var2;
    var1 = 2;
    var2 = 3;
    var1 = foo1(var1);
    var2 = foo1(var2);
    return 0;
}
    
```



Mohsan Jameel, Information Systems and Machine Learning Lab, University of Hildesheim

## Basic Execution Environment

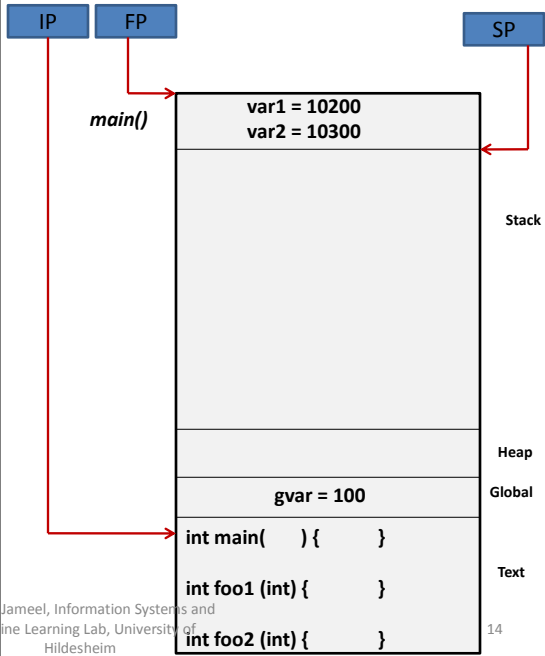
```

int gvar = 100;

int foo2 (int b)
{
    return b * gvar;
}

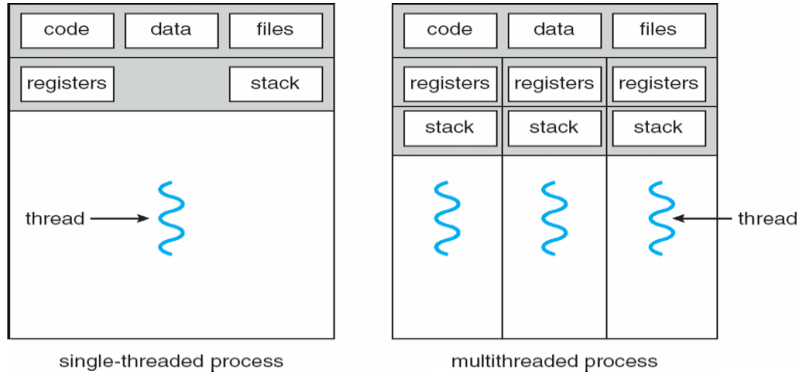
int foo1 (int a)
{
    int lvar = a + gvar;
    return foo2(lvar);
}

int main ( )
{
    int var1, int var2;
    var1 = 2;
    var2 = 3;
    var1 = foo1(var1);
    var2 = foo1(var2);
    return 0;
}
    
```



Mohsan Jameel, Information Systems and Machine Learning Lab, University of Hildesheim

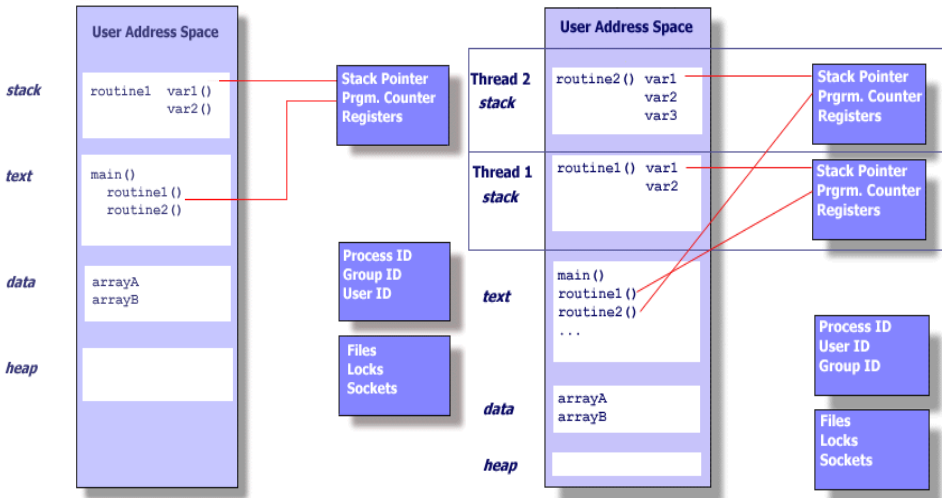
# What is a thread?



- ❑ What's needed to run code on CPU
  - ❖ "execution stream in an execution context"
  - ❖ Execution stream: sequential seq. of instructions
- ❑ CPU execution context (1 thread)
  - ❖ State: stack, heap, registers
  - ❖ Position: Instruction Pointer(IP) register
- ❑ OS execution context (n threads):
  - ❖ identity + open file descriptors, page table, ...

Mohsan Jameel, Information Systems and Machine Learning Lab, University of Hildesheim

# What is a thread?



Mohsan Jameel, Information Systems and Machine Learning Lab, University of Hildesheim



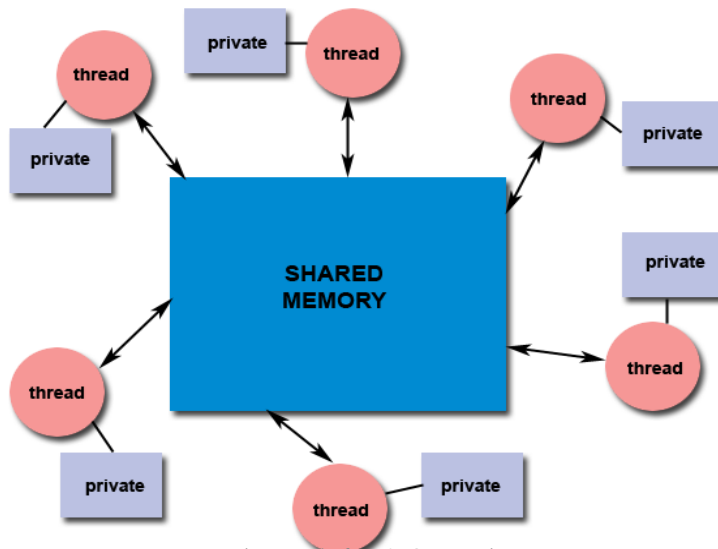
## What is a thread?

- ❑ All threads in a process share the same address space.
  - ❖ `*(char *)0xc000` means “the same” in thread T1 and T2.
- ❑ All threads share the same file descriptors.
  - ❖ Which implies that they share network sockets.
- ❑ All threads have access to the same heap and same global variables.
- ❑ Write access to global variables should be protected by a synchronization mechanism.
- ❑ Each thread has its separate stack, Instruction Pointer and Local variables.
  - ❖ Therefore each thread has its own independent flow of execution

Mohsan Jameel, Information Systems and  
Machine Learning Lab, University of  
Hildesheim

17

## What is a thread?



Mohsan Jameel, Information Systems and  
Machine Learning Lab, University of  
Hildesheim

18

# Java Threads

Mohsan Jameel, Information Systems and  
Machine Learning Lab, University of  
Hildesheim

19

## Thread class

Each thread is an object of the **Thread** class.

(Java tutorial says: “Each thread is associated with an instance of the class [Thread](#).”)

Java provide two basic ways to creates a thread:

1. Define a class that extends the class **Thread**.
2. Make your class implement the **Runnable** interface

Mohsan Jameel, Information Systems and  
Machine Learning Lab, University of  
Hildesheim

20

Simplest way is:

1. Define a class that extends the class `Thread`.

- Object of this class is a thread.
- Provide the method called `run` (which will override the inherited `run` method, which does nothing).
- The `run` method defines the code for the thread.
- Invoke the `start` method, which initiates the computation of the thread

Mohsan Jameel, Information Systems and  
Machine Learning Lab, University of  
Hildesheim

21

## Example

```
public class HelloThread extends Thread {
```

```
    public void run() {
        System.out.println("Hello from a thread!");
    }
```

```
    public static void main(String args[ ]) {
```

```
        HelloThread myThread = new HelloThread();
```

```
        myThread.start();
```

```
        System.out.println("Hello from the Main!");
```

```
        myThread.join();
```

```
    }
```

```
}
```

Java entry point

Create  
Thread  
object

Start thread and execute  
run method

Mohsan Jameel, Information Systems and  
Machine Learning Lab, University of  
Hildesheim

22

## Example 2

```

public class SimpleThread extends Thread {
    public SimpleThread(String str) { super(str); }
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println(i + " " + getName());
            try { // at this point, current thread is 'this'.
                Thread.sleep((long)(Math.random() * 1000));
            } catch (InterruptedException e) {}
        }
        System.out.println("DONE! " + getName());
    }
}

```

Mohsan Jameel, Information Systems and  
Machine Learning Lab, University of  
Hildesheim

23

```

public class TwoThreadsTest {
    public static void main (String[] args) {
        new SimpleThread("Thread1").start();
        new SimpleThread("Thread2").start();
    }
}

```

possible output

0 Thread1	5 Thread1	DONE! Thread2
0 Thread2	5 Thread2	9 Thread1
1 Thread2	6 Thread2	DONE! Thread1
1 Thread1	6 Thread1	
2 Thread1	7 Thread1	
2 Thread2	7 Thread2	
3 Thread2	8 Thread2	
3 Thread1	9 Thread2	
4 Thread1	8 Thread1	
4 Thread2		

Mohsan Jameel, Information Systems and  
Machine Learning Lab, University of  
Hildesheim

24

The **Thread** class actually implements the interface called **Runnable**.

The **Runnable** interface defines the single method, **run**, meant to contain the code executed in the thread.

Alternate more powerful way to create threads:

## 2. Make your class explicitly implement the **Runnable** interface

```
package java.lang;
public interface Runnable { public void run() ; }
```

Mohsan Jameel, Information Systems and  
Machine Learning Lab, University of  
Hildesheim

25

## Example

```
public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[ ]) {

        HelloRunnable myThread = new HelloRunnable();
        Thread tr = new Thread(myThread);
        tr.start();
        tr.join();
    }
}
```

Mohsan Jameel, Information Systems and  
Machine Learning Lab, University of  
Hildesheim

26

Runnable object can subclass a class other than Thread, i.e.:

```
public class MyRunnable extends SomeClass implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }
    public static void main(String args[ ]) {
        (new Thread(new HelloRunnable())).start();
    }
}
```

Note: both the Thread class and the Runnable interface are part of the standard Java libraries (java.lang package)

## Java's Thread class

Various instance and class methods, setters and getters:

- Class methods:
  - **sleep()**
  - ...
- Instance methods:
  - **join()**
  - **start()**
  - ...

<http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html>

# Race conditions and Synchronization

Mohsan Jameel, Information Systems and  
Machine Learning Lab, University of  
Hildesheim

29

## Race Conditions

- “A race condition is a programming fault producing undetermined program state and behavior due to unsynchronized parallel program executions” – [Liang Chen's Blog]
- **Therac-25 radiation therapy machine** — killed 3 people and seriously injured many more.
- **North American Blackout of 2003** — left 50 million people without power

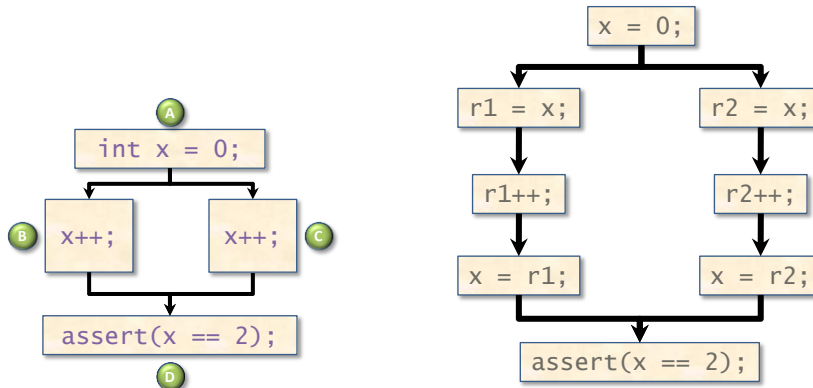
*Race bugs are notoriously  
difficult to discover by  
conventional testing!*

Mohsan Jameel, Information Systems and  
Machine Learning Lab, University of  
Hildesheim

30

## Determinacy race

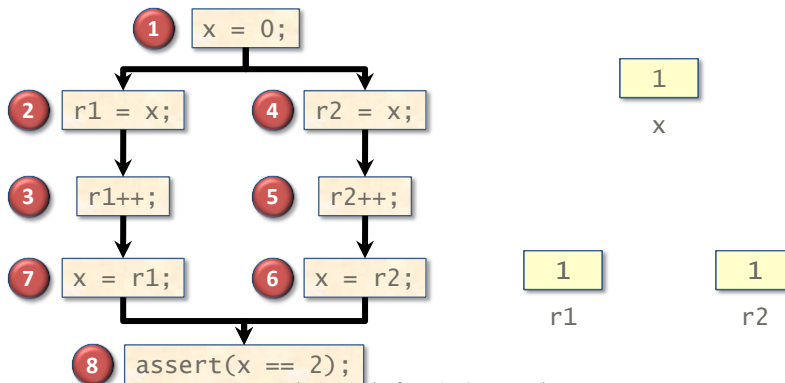
- Definition.** A *determinacy race* occurs when two logically parallel instructions access the same memory location and at least one of the instructions performs a write.



Mohsan Jameel, Information Systems and  
Machine Learning Lab, University of  
Hildesheim

31

## Example



Mohsan Jameel, Information Systems and  
Machine Learning Lab, University of  
Hildesheim

32



## Types of Races

- For given X and Y instructions, suppose both update a memory location A. Then following scenario could occur.

X	Y	Race Type
read	read	none
read	write	read race
write	read	read race
write	write	write race

Mohsan Jameel, Information Systems and  
Machine Learning Lab, University of  
Hildesheim

33

## Java Synchronization

- Java provides synchronized keyword to synchronize blocks of instructions. It can be used with:
  - a block of code
  - to a method body.
- The thread first arrived at the synchronized keyword acquire lock and rest of the threads arriving later are blocked.
  - Once a lock is released, only one of the waiting thread get the lock.

### Example

```
public class CounterClass {
    private int counter = 0;
    public synchronized void increment() {
        counter ++;
    }
    public synchronized void decrement() {
        counter --;
    }
    public synchronized int getValue() {
        return counter;
    }
}
```

Mohsan Jameel, Information Systems and  
Machine Learning Lab, University of  
Hildesheim

34

## Example using synchronized methods

### On-line banking

Several entities can access account potentially simultaneously (maybe a joint account, maybe automatic debits, ...)

Suppose three entities each trying to perform an operation, either:

- deposit()
- withdraw()
- enquire()

Mohsan Jameel, Information Systems and  
Machine Learning Lab, University of  
Hildesheim

35

### Create three threads, one for each entities

```
class InternetBankingSystem {
    public static void main(String [] args ) {
        Account accountObject = new Account ();
        Thread t1 = new Thread(new MyThread(accountObject));
        Thread t2 = new Thread(new YourThread(accountObject));
        Thread t3 = new Thread(new HerThread(accountObject));

        t1.start();
        t2.start();
        t3.start();

        // DO some other operation
        t1.join();
        t2.join();
        t3.join();

    } // end main()
}
```

Mohsan Jameel, Information Systems and  
Machine Learning Lab, University of  
Hildesheim

36

## Shared account

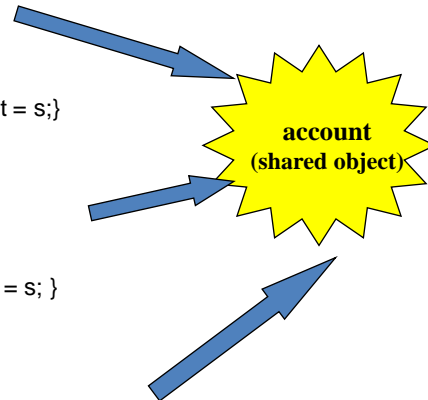
```

class MyThread implements Runnable {
    Account account;
    public MyThread (Account s) { account = s;}
    public void run() { account.deposit(); }
} // end class MyThread

class YourThread implements Runnable {
    Account account;
    public YourThread (Account s) { account = s;}
    public void run() { account.withdraw(); }
} // end class YourThread

class HerThread implements Runnable {
    Account account;
    public HerThread (Account s) { account = s; }
    public void run() {account.enquire(); }
} // end class HerThread

```



Mohsan Jameel, Information Systems and  
Machine Learning Lab, University of  
Hildesheim

37

## Synchronized account methods

```

class Account {
    int balance;
    // if 'synchronized' is removed, outcome unpredictable

    public synchronized void deposit( ) {
        balance += deposit_amount;
    }

    public synchronized void withdraw( ) {
        balance -= deposit_amount;
    }
    public synchronized void enquire( ) {
        display balance.
    }
}

```

Mohsan Jameel, Information Systems and  
Machine Learning Lab, University of  
Hildesheim

38

## Synchronized Statements

Unlike synchronized methods, synchronized statements must specify the object that provides the intrinsic lock:

Uses construction:

```
synchronized ( expression ) {
    statements
}
```

Evaluate to an object or an array. Used to identify lock.

“critical section”

Mohsan Jameel, Information Systems and  
Machine Learning Lab, University of  
Hildesheim

39

## Synchronized Statements

### Example

```
void incCount(){
    synchronized(this){
        count3++;
    }
}
```

Only this part synchronized

Mohsan Jameel, Information Systems and  
Machine Learning Lab, University of  
Hildesheim

40

## atomic action

An atomic action cannot stop in the middle: it either happens completely, or it doesn't happen at all. No side effects of an atomic action are visible until the action is complete.

Read/writes can be declared atomic with the **volatile** keyword, e.g.

**private volatile int x;**

Generally for smaller set of instruction it can be more efficient than synchronized methods.

## Coordinating threads Wait/notify mechanism

Sometimes need a thread to stop running and wait for an event before continuing.

**wait()** and **notify()** methods are methods of class Object.

Every object can maintain a list of waiting threads.

**wait()** When a thread calls **wait()** method of an object, any locks the thread holds are temporarily released and thread added to list of waiting threads for that object and stops running.

**notify()** When another thread calls **notify()** method on the same object, object wakes up one of the waiting threads and allows it to continue.

# References

- [Parallel Computing](#)
  - [https://computing.llnl.gov/tutorials/parallel\\_comp/](https://computing.llnl.gov/tutorials/parallel_comp/)
- [Java Threads](#)
  - <https://docs.oracle.com/javase/tutorial/essential/concurrency/runthead.html>
  - <http://www.wideskills.com/java-tutorial/java-threads-tutorial>
  - <https://computing.llnl.gov/tutorials/pthreads>
  - <http://se.inf.ethz.ch/old/teaching/ss2007/0284/book/Threads.pdf>