Advanced Java Programming

Naveen Kumar K.S

Adith.naveen@gmail.com

Objectives

- To introduce
  - **Concepts of Threads**
  - **Creating and managing threads**
  - **Priority Threads**
  - **Avoiding Race Conditions(synchronized)**

Multithreading

What are Threads?

- A thread is **not** an object
- A thread is a flow of control
- A thread is a series of executed statements
- A thread is a nested sequence of method calls
- Helps in introducing software parallelism

Multithreading	Multitasking
It is a programming concept	It is a system concept
It supports execution of multiple parts of a single program simultaneously	It supports executing of multiple programs simultaneously
The processor has to switch between different parts or threads of a program	The processor has to switch between different programs or processes
It is highly efficient	It is less efficient compared to multi threading
A thread is the smallest unit in multithreading	A program or process is the smallest unit in a multitasking environment
It helps in developing efficient programs	It helps in developing efficient operating system
It is cost-effective in case of context switching	It is expensive in case of context switching

How Threads are useful?

Multithreaded applications are most prevalent today

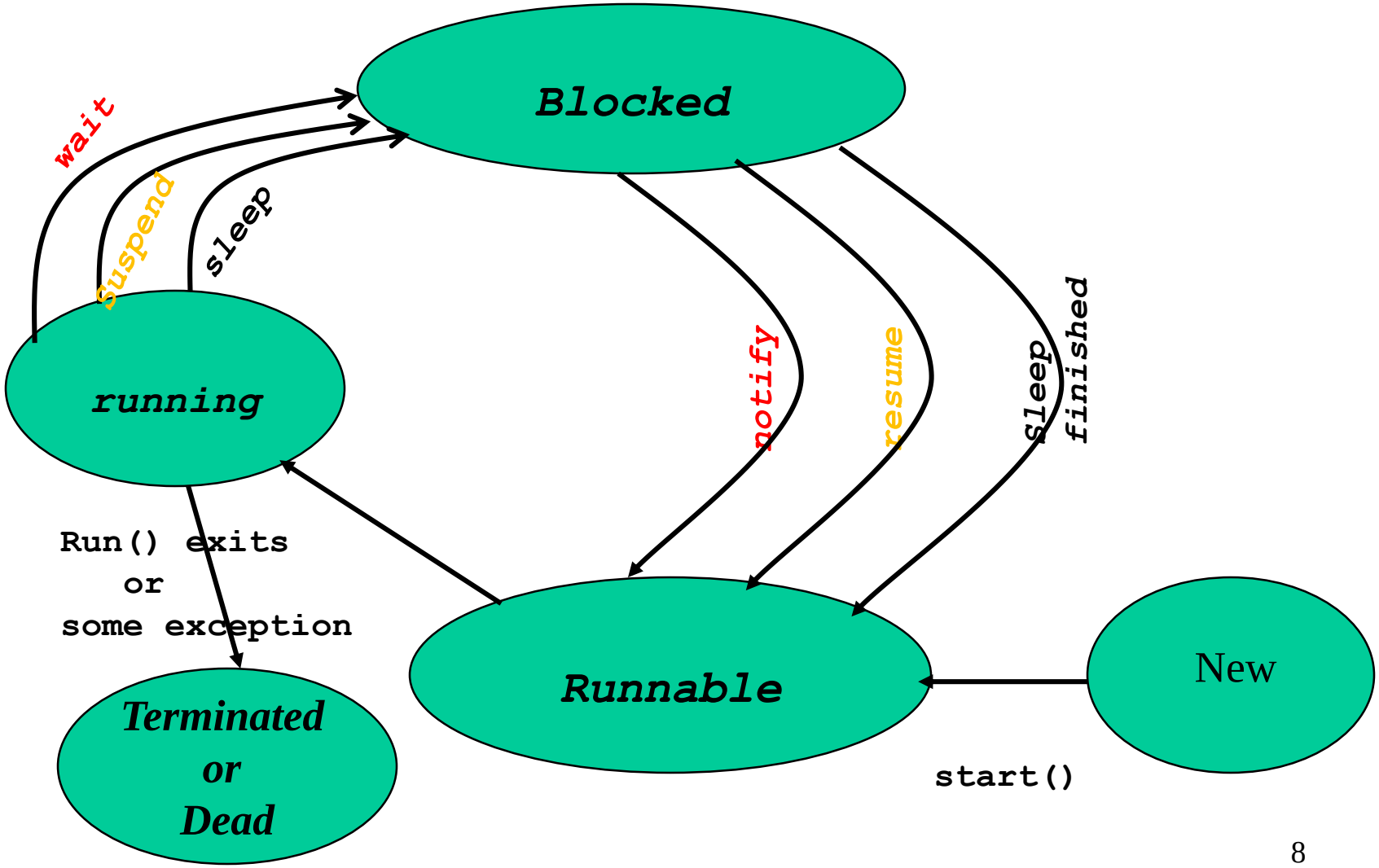
- Better utilization of system resources
- Multiple threads solve numerous problems better

Libraries of classes for programming multithreaded applications are available

Threads and Java

- All Java programs are threaded, may be implicitly
- Threading systems depend on the implementation on that platform

Thread States



Creating the Thread

There are two ways to create our own **Thread** object

1. Subclassing the **Thread** class and instantiating a new object of that class
2. Implementing the **Runnable** interface

In both cases the **run()** method should be implemented

The "Thread" class

- By subclassing the Thread class
- Other methods of Thread class can also be used

`void start()`

Creates a new thread and makes it runnable

`void run()`

The new thread begins its life inside this method

The "Runnable" Interface

- Implemented by classes whose instances are intended to be executed by a thread
- Need to implement the **`run()`** method
- Create a thread object using your **`Runnable`** object to perform thread operations

Starting the Thread

- Using the start( ) method
- Placing the thread in runnable state

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

Object A

```
Thread t = new BThread();

t.start();

doMoreStuff();
```

Object BThread (extends Thread)

```
BThread() {
}

void start() {
        // create thread
}

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

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

Object A

```
Thread t = new BThread();

t.start();

doMoreStuff();
```

Object BThread (extends Thread)

```
BThread() {
}

void start() {
    // create thread
}

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

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

Object A

```
Thread t = new BThread();

        t.start();

      doMoreStuff();
```

Object BThread (extends Thread)

```
BThread() {
}

void start() {
        // create thread
}

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

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

Object A

```
Thread t = new BThread();

        t.start();

      doMoreStuff();
```

Object BThread (extends Thread)

```
BThread() {
}

void start() {
        // create thread
}

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

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 Operations

- **sleep()**
- **wait()**
- **notify()**
- **stop()**

Race Conditions

- In multithreaded environment
- Two threads simultaneously contend for the same object
- Could result in an undefined state of the object, operated on
- Use of Java's **`synchronized`** keyword avoids these problems
- Implemented within the language

Synchronized -Putting it Together

- All access to delicate data should be `synchronized`.

Inter-thread Communications

- Threads talk to each other
- Threads wait for each other
- Two ways of communication:
  - through shared data
  - through thread-control methods

Thread Priorities

- Provides ten priority levels for threads
- Maps to the native OS priorities
  - In NT there are 7 levels
  - In Solaris, there are 2^{31} levels
- Use defined constants to set priorities:

  `MAX_PRIORITY, NORM_PRIORITY, MIN_PRIORITY`

Thank you….!!!!