НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Java language

Multitasking. Threads

Yevhen Berkunskyi, NUoS
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Multitasking

Multitasking allows several activities to occur concurrently on the computer. A distinction is usually made between:

- Process-based multitasking
- Thread-based multitasking

Multitasking

Some advantages of thread-based multitasking as compared to process-based multitasking are:

- threads share the same address space
- context switching between threads is usually less expensive than between processes
- the cost of communication between threads is relatively low

Overview of Threads

- Every thread in Java is created and controlled by a unique object of the java.lang.Thread class.
- Often the thread and its associated Thread object are thought of as being synonymous.

Threads make the runtime environment asynchronous, allowing different tasks to be performed concurrently.

Using this powerful paradigm in Java centers around understanding the following aspects of multithreaded programming:

- creating threads and providing the code that gets executed by a thread
- accessing common data and code through synchronization
- transitioning between thread states

The Main Thread

- When a standalone application is run, a user thread is automatically created to execute the main() method of the application. This thread is called the main thread.
- If no other user threads are spawned, the program terminates when the main() method finishes executing.

The Main Thread

All other threads, called child threads, are spawned from the main thread, inheriting its user-thread status. The main() method can then finish, but the program will keep running until all user threads have completed.

- Calling the setDaemon(boolean) method in the Thread class marks the status of the thread as either daemon or user, but this must be done before the thread is started.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

The Main Thread

When a GUI application is started, a special thread is automatically created to monitor the user–GUI interaction.

This user thread keeps the program running, allowing interaction between the user and the GUI, even though the main thread might have completed after the main() method finished executing.

Thread Creation

A thread in Java is represented by an object of the Thread class. Implementing threads is achieved in one of two ways:

- implementing the java.lang.Runnable interface
- extending the java.lang.Thread class

The Runnable interface has the following specification, comprising one abstract method declaration:

```
public interface Runnable {
    void run();
}
```

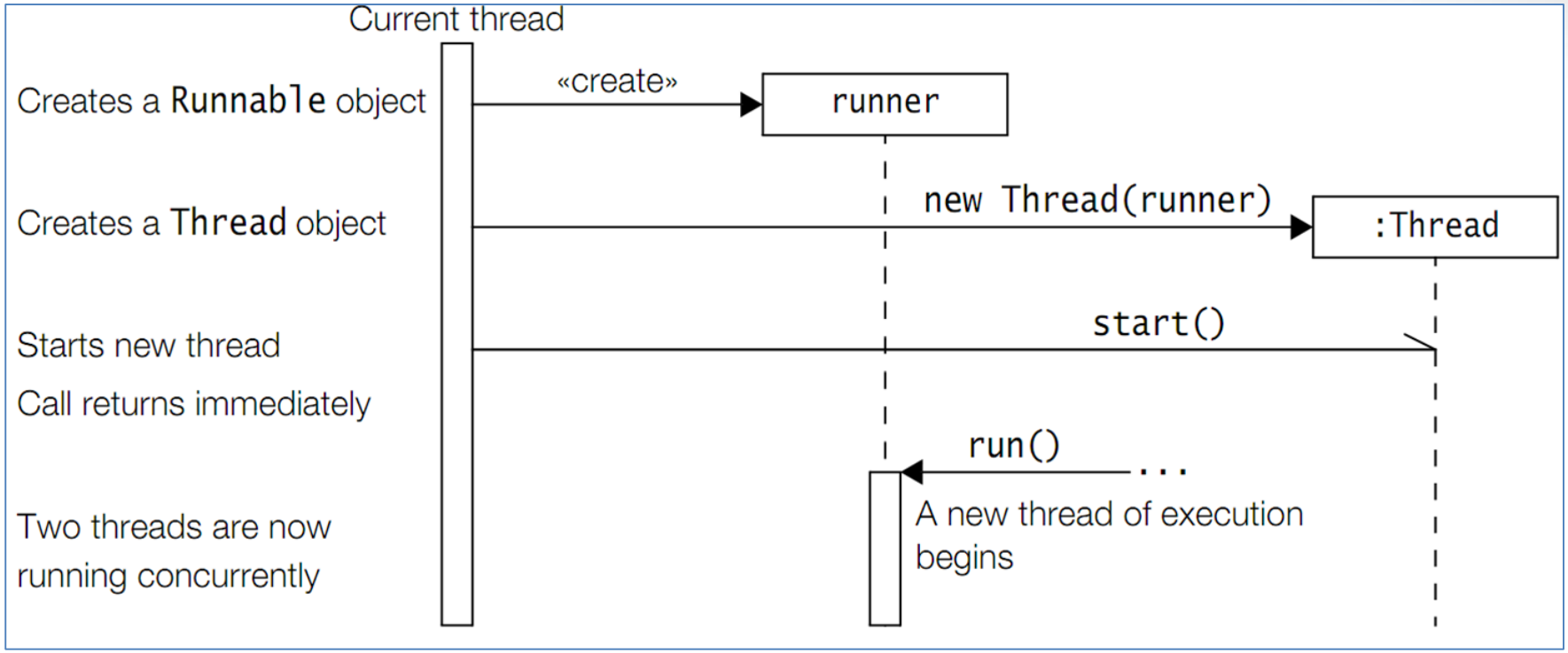
Implementing the Runnable Interface

The procedure for creating threads based on the Runnable interface is as follows:

1. A class implements the Runnable interface, providing the run() method that will be executed by the thread. An object of this class is a Runnable object.
2. An object of the Thread class is created by passing a Runnable object as an argument in the Thread constructor call. The Thread object now has a Runnable object that implements the run() method.
3. The start() method is invoked on the Thread object created in the previous step. The start() method returns immediately after a thread has been spawned. In other words, the call to the start() method is asynchronous.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Implementing the Runnable Interface



НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Important constructors and methods of java.lang.Thread class

```
Thread(Runnable target)
Thread(Runnable target, String threadName)
```

The argument target is the object whose run() method will be executed when the thread is started. The argument threadName can be specified to give an explicit name for the thread, rather than an automatically generated one. A thread's name can be retrieved by calling the getName() method.

```
static Thread currentThread()
```

This method returns a reference to the Thread object of the currently executing thread.

```
final String getName()
final void setName(String name)
```

The first method returns the name of the thread. The second one sets the thread's name to the specified argument.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Important constructors and methods of java.lang.Thread class

`void run()`

The Thread class implements the Runnable interface by providing an implementation of the run() method. This implementation in the Thread class does nothing and returns. Subclasses of the Thread class should override this method. If the current thread is created using a separate Runnable object, the run() method of the Runnable object is called.

`final void setDaemon(boolean flag)`

`final boolean isDaemon()`

The first method sets the status of the thread either as a daemon thread or as a user thread, depending on whether the argument is true or false, respectively. The status should be set before the thread is started. The second method returns true if the thread is a daemon thread, otherwise, false.

`void start()`

This method spawns a new thread, i.e., the new thread will begin execution as a child thread of the current thread. The spawning is done asynchronously as the call to this method returns immediately. It throws an IllegalThreadStateException if the thread is already started.

Extending the Thread Class

A class can also extend the Thread class to create a thread. A typical procedure for doing this is as follows:

1. A class extending the Thread class overrides the run() method from the Thread class to define the code executed by the thread.
2. This subclass may call a Thread constructor explicitly in its constructors to initialize the thread, using the super() call.
3. The start() method inherited from the Thread class is invoked on the object of the class to make the thread eligible for running.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

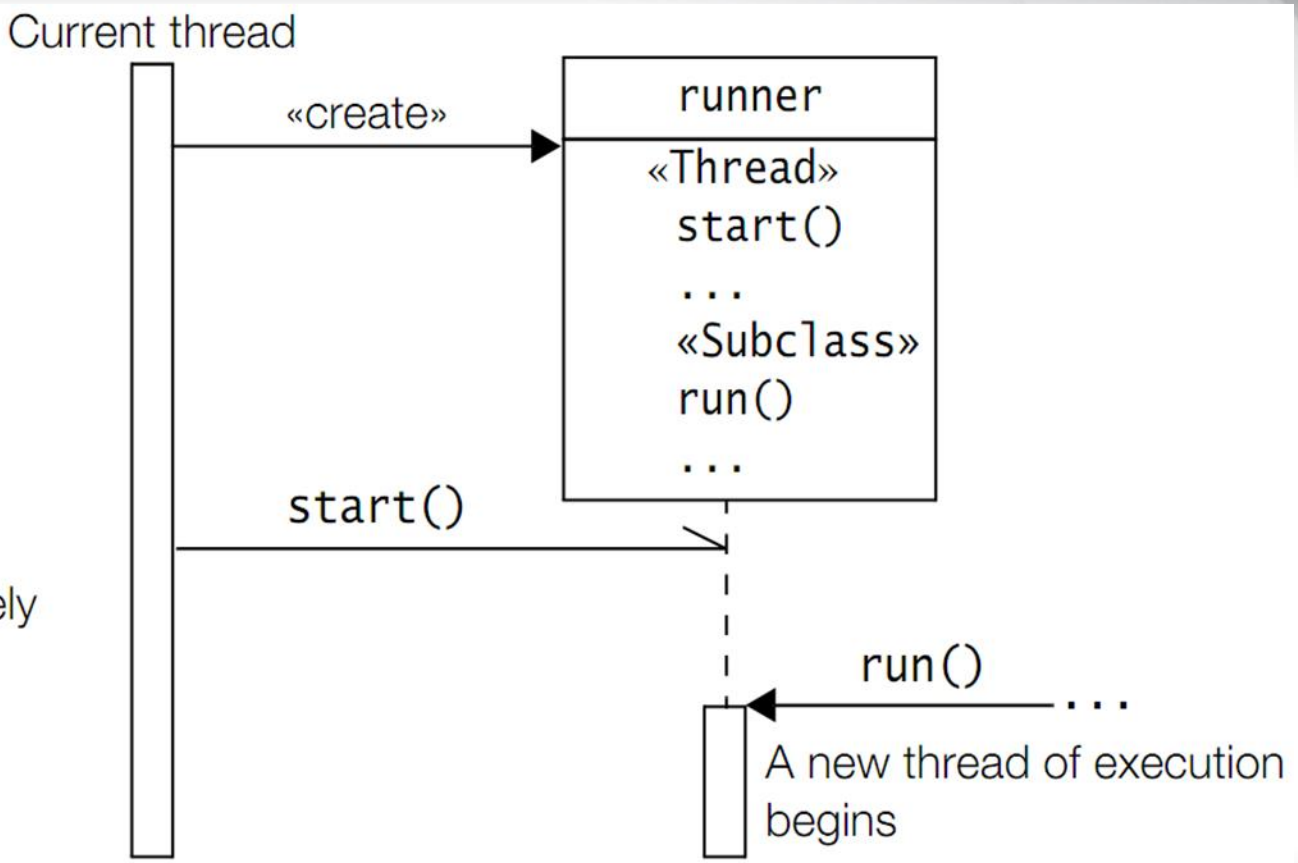
Extending the Thread Class

Creates an object of a class that extends the **Thread** class

Starts new thread

Call returns immediately

Two threads are now running concurrently



Synchronization

- Threads share the same memory space, i.e., they can share resources.
- However, there are critical situations where it is desirable that only one thread at a time has access to a shared resource.
- A lock (also called a monitor) is used to synchronize access to a shared resource. A lock can be associated with a shared resource.
- Threads gain access to a shared resource by first acquiring the lock associated with the resource. At any given time, at most one thread can hold the lock and thereby have access to the shared resource.
- In Java, all objects have a lock—including arrays. This means that the lock from any Java object can be used to implement mutual exclusion.

Synchronized Methods

- If the methods of an object should only be executed by one thread at a time, then the declaration of all such methods should be specified with the keyword synchronized.
- A thread wishing to execute a synchronized method must first obtain the object's lock (i.e., hold the lock) before it can enter the object to execute the method. This is simply achieved by calling the method.
- If the lock is already held by another thread, the calling thread waits. No particular action on the part of the program is necessary.
- A thread relinquishes the lock simply by returning from the synchronized method, allowing the next thread waiting for this lock to proceed.

Synchronized Methods

- While a thread is inside a synchronized method of an object, all other threads that wish to execute this synchronized method or any other synchronized method of the object will have to wait.
- This restriction does not apply to the thread that already has the lock and is executing a synchronized method of the object. Such a method can invoke other synchronized methods of the object without being blocked.
- The non-synchronized methods of the object can always be called at any time by any thread.

Synchronized Blocks

Whereas execution of synchronized methods of an object is synchronized on the lock of the object, the synchronized block allows execution of arbitrary code to be synchronized on the lock of an arbitrary object. The general form of the synchronized statement is as follows:

```
synchronized(<object reference expression>){
    <code block>
}
```

Synchronized Blocks

- The <object reference expression> must evaluate to a non-null reference value, otherwise a NullPointerException is thrown.
- The code block is usually related to the object on which the synchronization is being done.
- This is analogous to a synchronized method, where the execution of the method is synchronized on the lock of the current object

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Example

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Questions?

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Java language

Multitasking. Threads

Yevhen Berkunskyi, NUoS
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua