

НИШКИ

ЛЕКЦИОНЕН КУРС “ПРОГРАМИРАНЕ НА JAVA”



СТРУКТУРА НА ЛЕКЦИЯТА

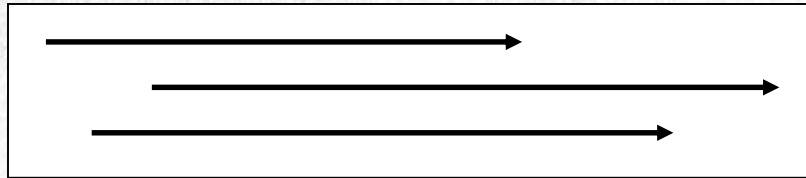
- Паралелност
- Видове паралелност
- Работа с нишки
- Жизнен цикъл
- Работа с нишки

НЯМА ИЗПОЛЗВАНЕ НА КОМПЮТЪРА БЕЗ ПАРАЛЕЛНОСТ

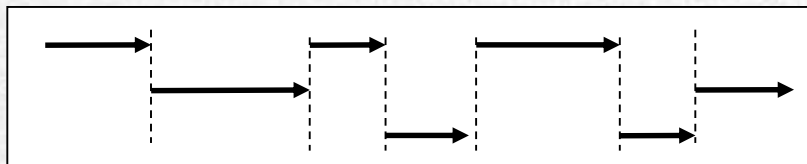
- Използване ресурсите на компютъра:
много неща в него се извършват паралелно
→ печат, вход от клавиатура, анимация, ...
- Между натискане на два клавиша:
милиони машинни операции
- Windows XP: около 80 фонове услуги
→ автоматично зареждане на updates от Microsoft-Servers,
управление на печата, мрежови услуги (напр.
установяване връзка със сървър), създаване backups на
системни файлове, комуникация на програми и
операционни системи, скенери за вируси, генериране на
протоколи за грешки ...

ПАРАЛЕЛНОСТ

- Повече програми работят едновременно



- Ако компютърът е с един процесор: също така възможност за псевдопаралелност
→ изчислителното време разпределено поинтервално между програмите



Процес:
последователна
програма

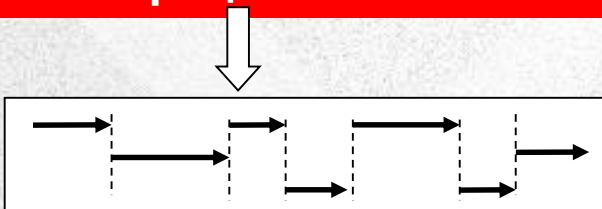
- Предимство: използват се времената на изчакване на другите програми

ПРОЦЕСИ

- Програмни единици, които се изпълняват независимо от останалите части на програмата
- Видове процеси
 - Леки
 - Тежки

КОНТРОЛ НА ПАРАЛЕЛНОСТ

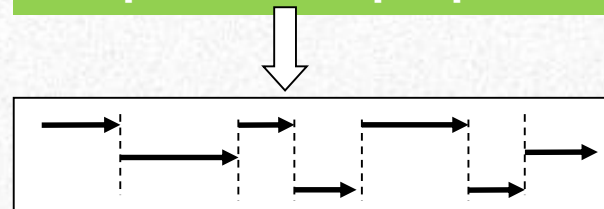
Операционна система



„Тежки“ процеси:

- всеки процес със собствена памет
- паметта е защитена от достъп на други процеси
- комуникацията тежка: обмен на съобщения през операционната система

Приложна програма



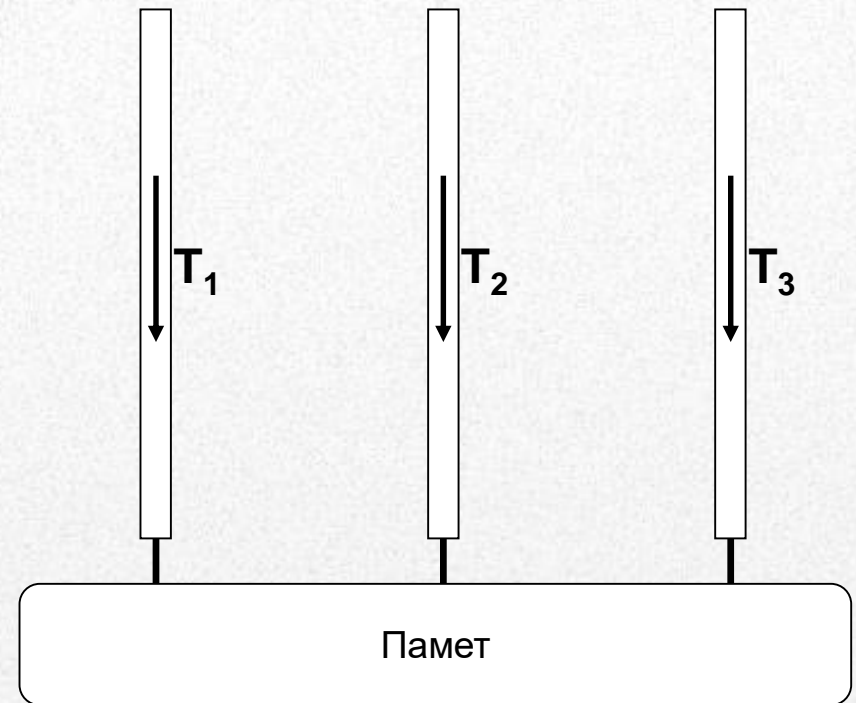
„Леки“ процеси (Threads):

- комуникация посредством обща ОП
- по-ефективни
- по-несигурни

ПРОЦЕСИ & THREADS



Тежки процеси

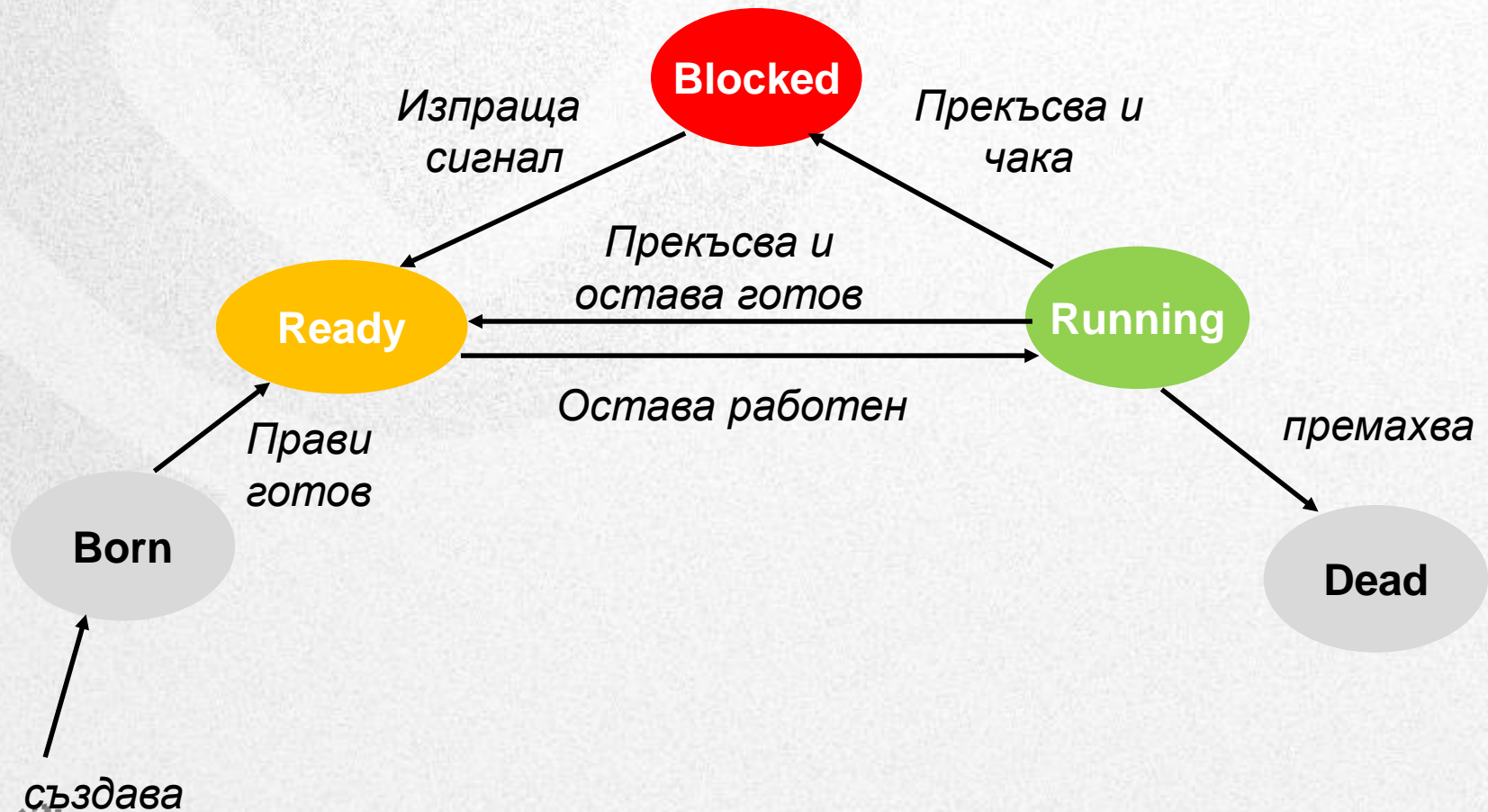


Леки процеси (Threads)

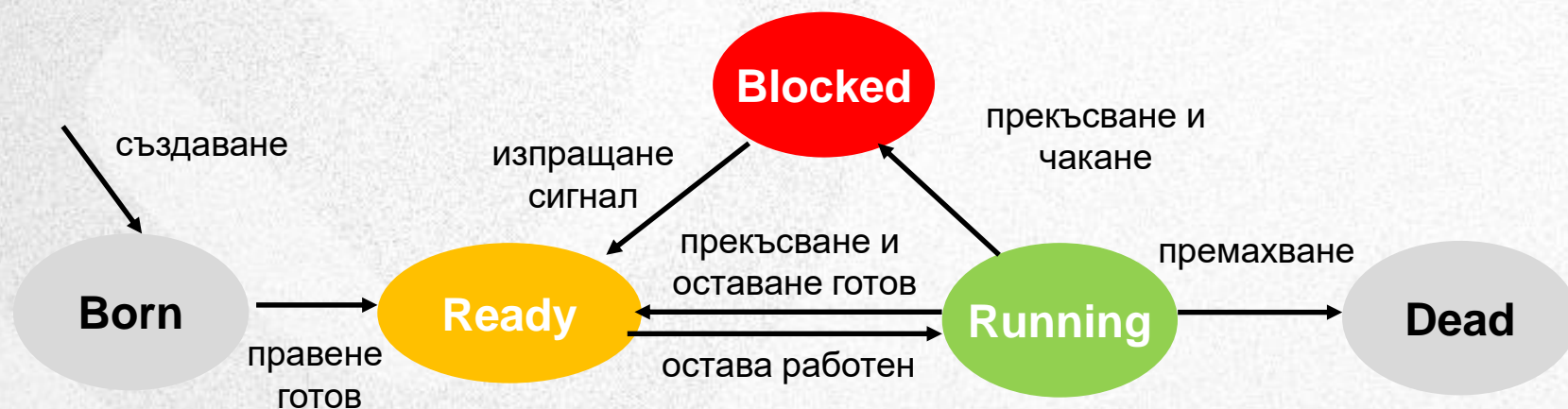
РАБОТА С НИШКИ

- Реализация на клас, разширяващ класа **Thread**
- Разполагане на кода на нашата задача в **run-**метода на подкласа
- Създаване на обект на подкласа
- Стартиране на нишката със **start**-метода

ЖИЗНЕН ЦИКЪЛ НА THREADS: МОДЕЛ НА СЪСТОЯНИЯ



ПРЕХОДИ НА СЪСТОЯНИЯТА В ДЕТАЙЛИ:СЪЗДАВАНЕ И ПРАВЕНЕ ГОТОВ



Създаване thread:

```
Thread t = new Thread();
```

Born

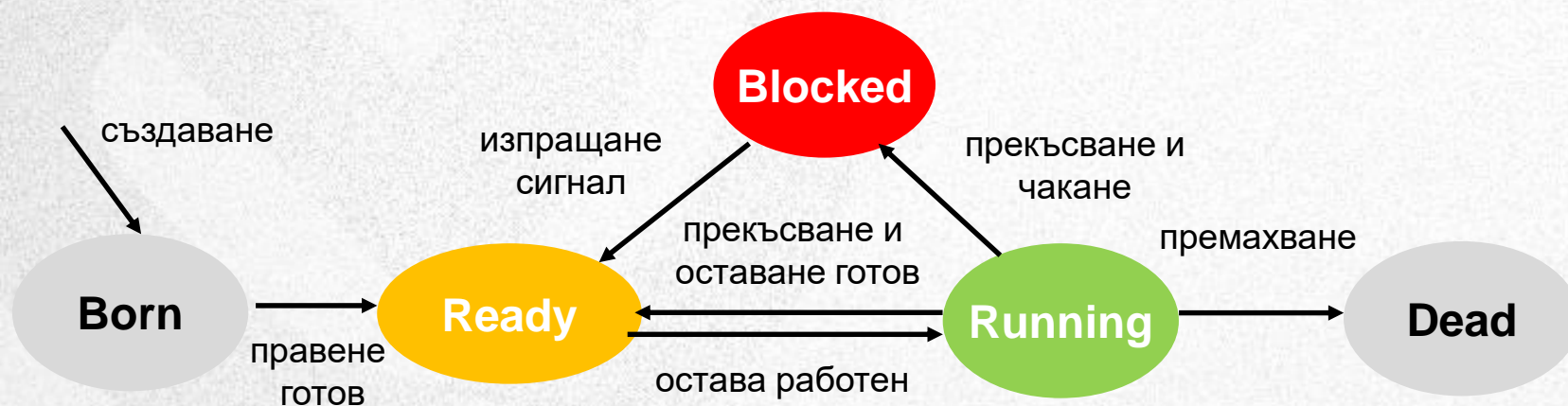
Правене готов – Thread стартиране:
друг активен Thread стартира новия Thread t

```
t.start();
```

Born

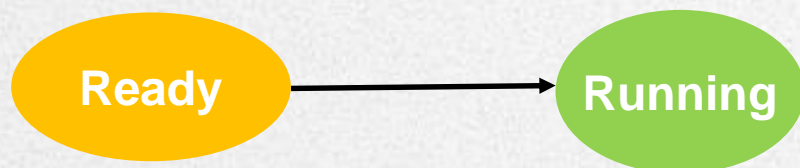
Ready

ПРЕХОДИ НА СЪСТОЯНИЯТА В ДЕТАЙЛИ: ОСТАВА РАБОТЕН И ПРЕКЪСВАНЕ



Thread остава работен:

Java-VM:
Scheduler решава

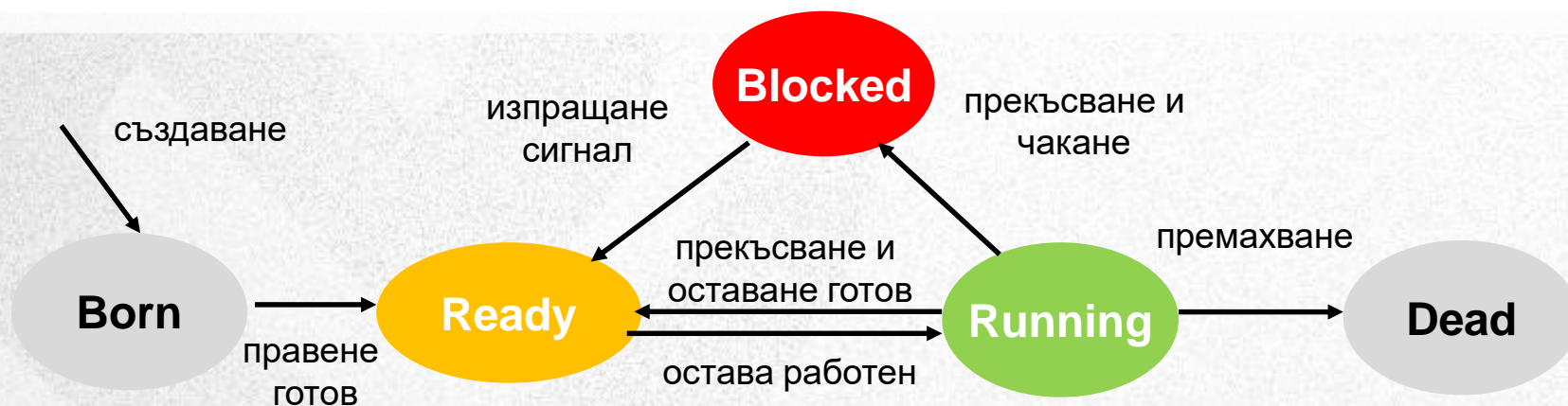


Thread прекъсване и остава готов:

- изразходва времеви интервал
- смяна с процес с по-висок приоритет
- процес освобожда доброволно процес: `t.yield()`



ПРЕХОДИ НА СЪСТОЯНИЯТА В ДЕТАЙЛИ: ПРЕКЪСВАНЕ И ЧАКАНЕ



Прекъсване и чакане:

- самозамразява се: `t.sleep()`
- чака за края на друг Thread `u`: `u.join()`
- чака за разблокиране на обект `o`: `o.wait()`

Получава сигнал и отново готов:

- времето за замразяване завършва
- събужда друг Thread `t`: `t.interrupt()`
- друг Thread, края на който чака, е „dead“
- обект `o` е наличен



API-CLASS: THREAD (PART)

[java.lang.Object](#)
↳ [java.lang.Thread](#)

All Implemented Interfaces:
[Runnable](#)

```
public class Thread
extends Object
implements Runnable
```

A *thread* is a thread of execution in a program. The Java Virtual Machine

Every thread has a priority. Threads with higher priority are executed in greater order. If a new thread object, the new thread has its priority initially set equal to the priority of the thread that created it.

Since:
JDK1.0

See Also:
[Runnable](#), [Runtime.exit\(int\)](#), [run\(\)](#), [stop\(\)](#)

**A *thread* is a thread of execution in a program.
The Java Virtual Machine allows an application
to have multiple threads of execution running concurrently.
Every thread has a priority ...**

as a

Constructor Summary

[Thread\(\)](#)
Allocates a new Thread object.

[Thread\(Runnable target\)](#)
Allocates a new Thread object.

Method Summary

static Thread	currentThread()	Returns the current thread.
void	interrupt()	Interrupts the thread.
boolean	isAlive()	Tests if the thread is alive.
boolean	isInterrupted()	Tests if the thread has been interrupted.
void	join()	Waits for the thread to die.
void	join(long millis)	Waits for the thread to die, with a timeout.
void	run()	Executes the thread.
void	setPriority(int newPriority)	Changes the priority of the thread.
static void	sleep(long millis)	Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
static void	yield()	Causes the currently executing thread object to temporarily pause and allow other threads to execute.

Избор:

```
static Thread currentThread()
void interrupt()
boolean isAlive()
boolean isInterrupted()
void join()
void join(long millis)
void run()
void setPriority(int newPriority)
static void sleep(long millis)
static void yield()
```

Thread does nothing

СЪЩЕСТВЕНИ МЕТОДИ

```
class Thread implements Runnable {  
  
    void start()                Thread start  
    void run()                 Program of Thread  
    void interrupt()           wake  
    void join ()               wait the end of another Thread  
    void join (long millisec)  wait ... max millisec  
  
    boolean isAlive()          Thead active?  
  
    int getPriority()           Priority answer  
    void setPriority()          Priority set  
  
    static Thread currentThread()    actual Thread-Object  
  
    static void sleep (long milliseconds)  pause  
    static void yield()                    contorol pass on  
}
```


ПРОЦЕСИ КАТО ОБЕКТИ

Процеси (Threads):

- произхождат от областта на динамичните обработки
- алгоритми → императивно програмиране

Java: отношение към ООП

Модел: Към всеки Thread принадлежи един обект, който го контролира (Object = единица от данни и оператори ...)

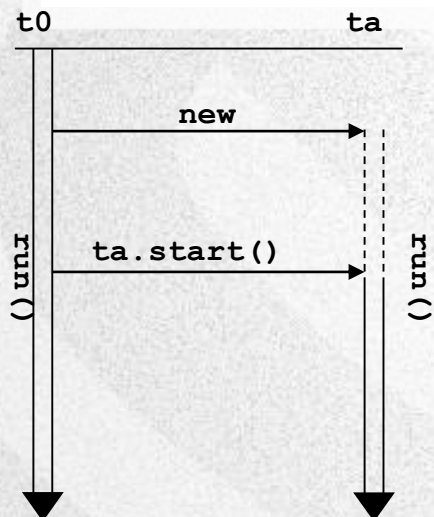
→ двата взаимно се идентифицират: „Thread t1“ – всъщност се има предвид обекта, който контролира Thread

Основен принцип: JVM стартира един „Great-Thread“ (базов процес), който изпълнява main()

РАБОТА С THREADS

- Създаване, стартиране, обработка, премахване
- Замразяване
- Изчакване
- Постановяне на приоритети

СЪЗДАВАНЕ, СТАРТИРАНЕ, ОБРАБОТКА, ПРЕМАХВАНЕ (1)



създаване: един работещ Thread създава нов `ta = new ThreadA1();`

стартиране: работещ Thread стартира нов `ta.start();`

обработка: JVM-Scheduler стартира `run()`

премахване: `run()` завършва обработката

```
class ThreadA1 extends Thread {
    public void run() {
        for (int i = 1; i < ThreadBasicTest.LIMIT; i++) {
            System.out.println("A: " + i);
        }
        System.out.println("A done");
    }
}
```


ДВА ТРИВИАЛНИ THREADS: ИЗВЕЖДАНЕ НА ЧИСЛА

```
public class ThreadBasicTest {
    static final int LIMIT = 10;
    public static Thread ta;
    public static Thread tb;

    public static void main(String[] args) {
        ta = new ThreadA1();
        tb = new ThreadB1();
        ta.start();
        tb.start();
        System.out.println("    done...");
    }
}
```

```
class ThreadA1 extends Thread {
    public void run() {
        for (int i = 1; i < ThreadBasicTest.LIMIT; i++) {
            System.out.println("A: " + i);
        }
        System.out.println("A done");
    }
}
```

```
class ThreadB1 extends Thread {
    public void run() {
        for (int i = -1; i > -ThreadBasicTest.LIMIT; i--) {
            System.out.println("\t\tB: " + i);
        }
        System.out.println("\t\tB done");
    }
}
```

РЕЗУЛТАТИ (TOSHIBA ЛАПТОП)

1

done...

A: 1

B: -1

B: -2

B: -3

B: -4

B: -5

B: -6

B: -7

B: -8

B: -9

B done

A: 2

A: 3

A: 4

A: 5

A: 6

A: 7

A: 8

A: 9

A done

2

done...

A: 1

A: 2

A: 3

A: 4

A: 5

A: 6

A: 7

A: 8

A: 9

A done

B: -1

B: -2

B: -3

B: -4

B: -5

B: -6

B: -7

B: -8

B: -9

B done

3

A: 1

A: 2

A: 3

A: 4

A: 5

A: 6

A: 7

A: 8

A: 9

A done

done...

B: -1

B: -2

B: -3

B: -4

B: -5

B: -6

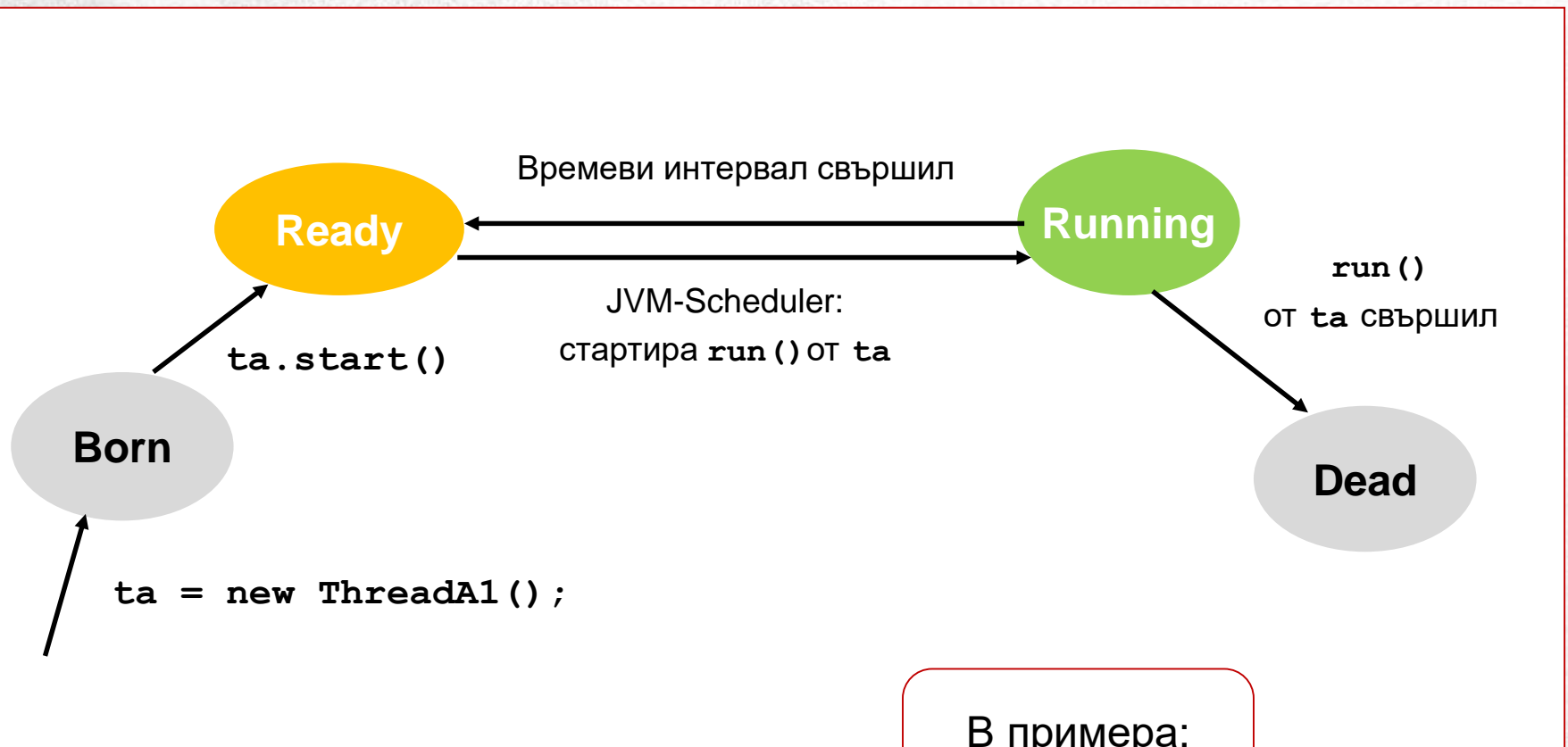
B: -7

B: -8

B: -9

B done

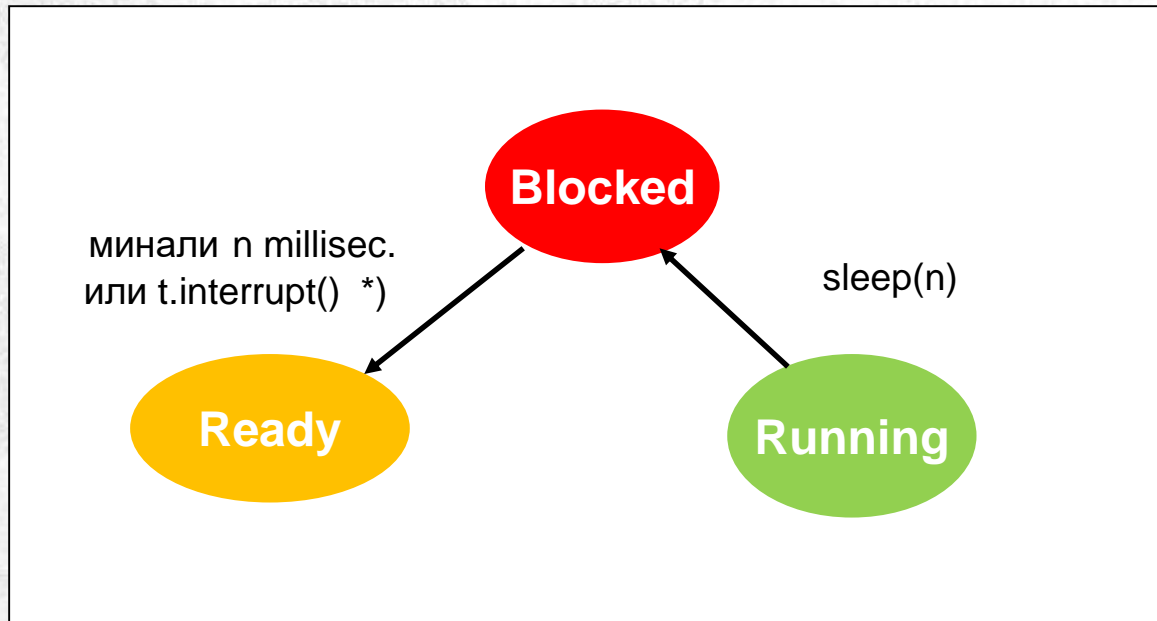
СЪЗДАВАНЕ, СТАРТИРАНЕ, ОБРАБОТКА, ПРЕМАХВАНЕ: ПРЕХОДИ НА СЪСТОЯНИЯТА



ПРИМЕРЪТ ДЕМОНСТРИРА СЪЩЕСТВЕНИ ХАРАКТЕРИСТИКИ НА НИШКИТЕ

- Thread scheduler не дава гаранция за реда, в който се изпълняват нишките
- Всяка нишка се обработва в един кратък времеви период, наречен **time slice**
- След което thread scheduler избира за активиране друга нишка от един пул с **runnable** нишки
- Една нишка е **runnable** ако в момента не спи или не е блокирана по някакъв начин

SLEEP: THREADS ПРЕКЪСВАТ САМИ РАБОТАТА СИ



***) друг Thread може да събуди t преди времето: `t.interrupt()`**

→ Произвежда изключение `InterruptedException`

→ `sleep` трябва да бъде вграден в `try-catch` (в противен случай: `Compiler error`)

THREAD-ПРИМЕР СЪС SLEEP()

```
class ThreadA2 extends Thread {  
  
    public void run() {  
        for (int i = 1; i < ThreadSleep.LIMIT; i++) {  
            try {  
                sleep(60);  
            } catch (InterruptedException e) {}  
            System.out.println("A: " + i);  
        }  
        System.out.println("A done");  
    }  
}
```

```
class ThreadB2 extends Thread {  
  
    public void run() {  
        for (int i = -1; i > -ThreadSleep.LIMIT; i--) {  
            try {  
                sleep(40);  
            } catch (InterruptedException e) {}  
            System.out.println("\t\tB: " + i);  
        }  
        System.out.println("\t\tB done");  
    }  
}
```

```
done...  
A: 1          B: -1  
A: 2          B: -2  
A: 3          B: -3  
A: 4          B: -4  
A: 5          B: -5  
A: 6          B: -6  
A: 7          B: -7  
A: 8          B: -8  
A: 9          B: -9  
A: 10         B: -10  
A: 11         B: -11  
A: 12         B: -12  
A: 13         B: -13  
A: 14         B: -14  
A: 15         B: -15  
A: 16         B: -16  
A: 17         B: -17  
A: 18         B: -18  
A: 19         B: -19  
A: 20         B: -20  
A done       B done
```


SPOTTEST: ЕДИН АПЛЕТ

```
public class SpotTest extends Applet {

    /* SpotTest      J M Bishop Aug 2000
     * =====
     *
     * Draws spots of different colours
     *
     * Illustrates simple threads
     */

    int mx, my;
    int radius = 10;
    int boardSize = 200;
    int change;

    public void init() {
        boardSize = getSize().width - 1;
        change = boardSize-radius;

        // creates and starts three threads
        new Spots(Color.red).start();
        new Spots(Color.blue).start();
        new Spots(Color.green).start();
    }
```

```
class Spots extends Thread {
    Color colour;

    Spots(Color c) {
        colour = c;
    }

    public void run () {
        while (true) {
            draw();
            try {
                sleep (500); // millisecs
            }
            catch (InterruptedException e) {}
        }
    }

    public void draw() {
        Graphics g = getGraphics();
        g.setColor(colour);
        // calculate a new place for a spot
        // and draw it.
        mx = (int) (Math.random()*1000) % change;
        my = (int) (Math.random()*1000) % change;
        g.fillOval(mx, my, radius, radius);
    }
}
```

SPOTTEST: АПЛЕТ С ПРИМЕР ЗА ИЗХОД

```
public class SpotTest extends Applet {
```

```
/* SpotTest      J M Bishop Aug 2000
```

```
* =====
```

```
*
```

```
* Draws spots of different c
```

```
*
```

```
* Illustrates simple threads
```

```
*/
```

```
int mx, my;
```

```
int radius = 10;
```

```
int boardSize = 200;
```

```
int change;
```

```
public void init() {
```

```
    boardSize = getSize().width;
```

```
    change = boardSize-radius;
```

```
    // creates and starts three
```

```
    new Spots(Color.red).start();
```

```
    new Spots(Color.blue).start();
```

```
    new Spots(Color.green).start();
```

```
}
```

```
class Spots extends Thread {  
    Color colour;
```

```
    Spots(Color c) {
```

```
        colour = c;
```

```
    } {
```

```
    {
```

```
    }; // millisecs
```

```
    InterruptedException e) {}
```

```
    } {
```

```
        getGraphics();
```

```
        colour);
```

```
        // calculate a new place for a spot
```

```
        // and draw it.
```

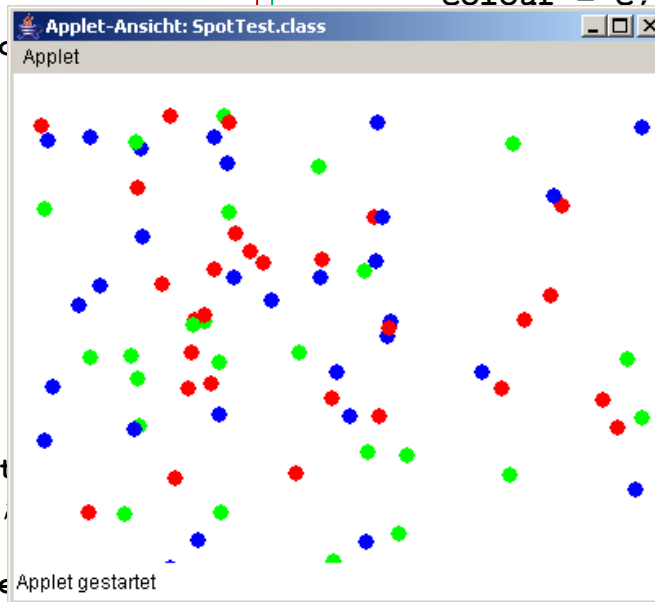
```
        mx = (int) (Math.random()*1000) % change;
```

```
        my = (int) (Math.random()*1000) % change;
```

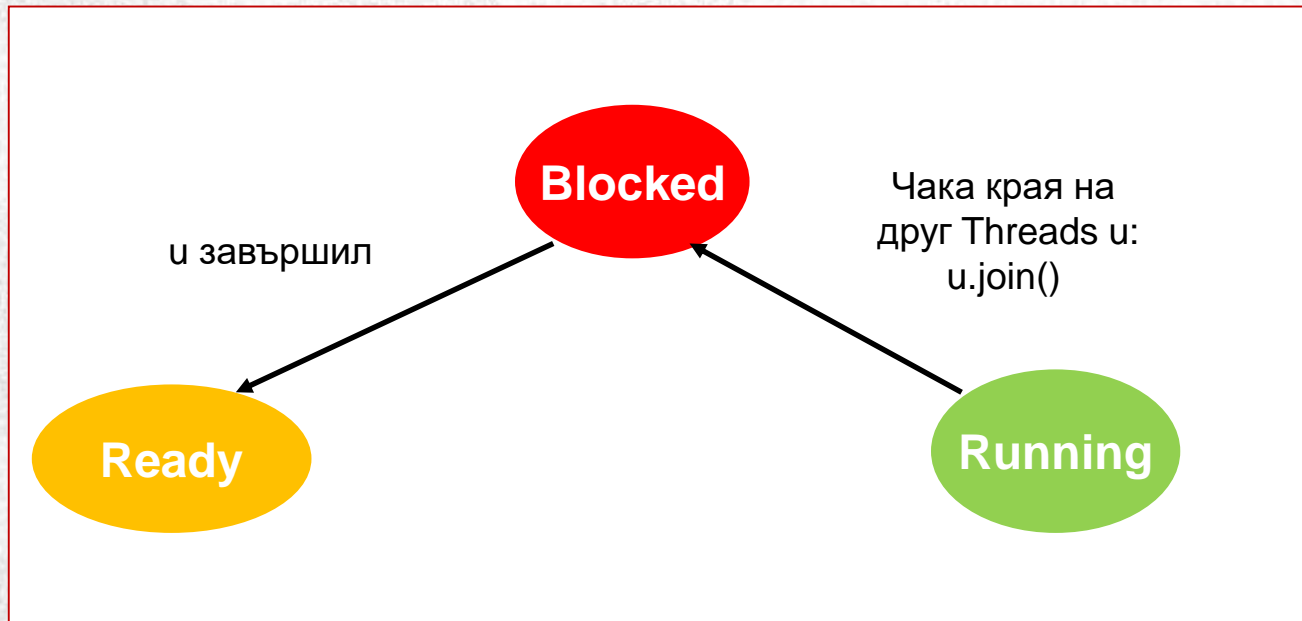
```
        g.fillOval(mx, my, radius, radius);
```

```
    }
```

```
}
```



JOIN: ЧАКА КРАЯ НА ДРУГ THREADS



Смислена по-нататъшна работа едва тогава възможна, когато обработката на *u* е завършила.

ПРИМЕР: СМИСЛЕНА ПО-НАТАТЪШНА РАБОТА ЕДВА СЛЕД КРАЯ НА ДРУГ THREADS

Чака края на процес ,Sorts‘

```
Sorts sort = new Sorts();  
.  
.  
sort.start(); //sorts large Array  
// next activities:  
.  
.  
sort.join(); //now: waiting the end of sort  
// now: access to sorted Array  
.  
.  
.
```

Процес ,Sorts‘

```
class Sorts extends Thread {  
    public void run() {  
        quicksort(...);  
    }  
}
```

THREAD-ПРИМЕР С JOIN()

```
class ThreadB3 extends Thread    {

    public void run()    {
        for (int i = -1; i > -ThreadJoin.LIMIT/2; i--){
            System.out.println("\t\tB: " + i);

            try {
                ThreadJoin.ta.join();
            }catch (InterruptedException e) {}
            System.out.println("\t\tB done");
        }
    }
}
```

done...

A: 1
A: 2
A: 3
A: 4
A: 5
A: 6
A: 7
A: 8

B: -1
B: -2
B: -3
B: -4
B: -5
B: -6
B: -7
B: -8
B: -9

A: 9
A: 10
A: 11
A: 12
A: 13
A: 14
A: 15
A: 16
A: 17
A: 18
A: 19
A: 20
A done

B done

ПОСТАВЯНЕ НА ПРИОРИТЕТИ: SETPRIORITY()

- Предпочитание към належащи задачи
- На всеки Thread се присвоява приоритет
- По-висок приоритет има предимство, ако два Threads са в състояние ,Ready‘
- Методи:
 - `getPriority()` пита
 - `setPriority()` поставя

THREADS С ПРИОРИТЕТИ

```
class ThreadB4 extends Thread    {

    public void run()    {
        for (int i = -1; i > -ThreadPriority.LIMIT; i--)    {
            System.out.println("\t\tB: " + i);
            if (i== -1)    {
                ThreadPriority.ta.setPriority(this.getPriority() + 1);
                System.out.println("Decreased");
            }
        }
        System.out.println("\t\tB done");
    }
}
```

done...

B: -1

Decreased

B: -2

B: -3

B: -4

B: -5

A: 1

A: 2

A: 3

A: 4

A: 5

A: 6

A: 7

A: 8

A: 9

A: 10

A: 11

A: 12

A: 13

A: 14

A: 15

A: 16

A: 17

A: 18

A: 19

A: 20

A done

B: -6

B: -7

B: -8

B: -9

B: -10

B: -11

B: -12

B: -13

B: -14

B: -15

B: -16

B: -17

B: -18

B: -19

B: -20

B done

THREADS С ПРИОРИТЕТИ: РАЗЛИЧНИ КОМПЮТРИ

```
class ThreadB4 extends Thread {  
  
    public void run() {  
        for (int i = -1; i > -ThreadPriority.LIMIT; i--)  
            System.out.println("\t\tB: " + i);  
        if (i == -1) {  
            ThreadPriority.ta.setPriority(this.getPriority());  
            System.out.println("Decreased");  
        }  
    }  
    System.out.println("\t\tB done");  
}
```

Очевидно: различна семантика

done...	B: -1
Decreased	
A: 1	
A: 2	
A: 3	
A: 4	
A: 5	
A: 6	
A: 7	
A: 8	
A: 9	
A: 10	
A: 11	
A: 12	
A: 13	
A: 14	
A: 15	
A: 16	
A: 17	
A: 18	
A: 19	
A: 20	
A done	
	B: -2
	B: -3
	B: -4
	B: -5
	B: -6
	B: -7
	B: -8
	B: -9
	B: -10
	B: -11
	B: -12
	B: -13
	B: -14
	B: -15
	B: -16
	B: -17
	B: -18
	B: -19
	B: -20
	B done

done...	B: -1
Decreased	
A: 1	
A: 2	
A: 3	
A: 4	
A: 5	
A: 6	
A: 7	
A: 8	
A: 9	
A: 10	
A: 11	
A: 12	
A: 13	
A: 14	
A: 15	
A: 16	
A: 17	
A: 18	
A: 19	
A: 20	
A done	
	B: -2
	B: -3
	B: -4
	B: -5
	B: -6
	B: -7
	B: -8
	B: -9
	B: -10
	B: -11
	B: -12
	B: -13
	B: -14
	B: -15
	B: -16
	B: -17
	B: -18
	B: -19
	B: -20
	B done

КОМУНИКАЦИЯ МЕЖДУ THREADS: ОБЩА ПАМЕТ = ОБЩИ ОБЕКТИ

Сметка:

```
class Account {  
    private long balance;  
  
    void deposit (long amount) {  
        long aux = this.balance;  
        aux = aux + amount;  
        this.balance = aux;  
    }  
  
    void withdraw (long amount) {  
        long aux = this.balance;  
        if (aux >= amount) {  
            aux = aux - amount;  
            this.balance = aux;  
        }  
    }  
}  
  
Account acc = new . . . ;
```

Threads: Различни клиенти на сметката

- клиент на автомата
- банков служител
- пълномощно за теглене

Клиент:
`acc.deposit(200)`

Пълномощно:
`acc.withdraw(200)`

КОМУНИКАЦИЯ МЕЖДУ THREADS: СИНХРОНИЗАЦИОНЕН ПРОБЛЕМ

Сметка:

```
class Account {  
    private long balance;  
  
    void deposit (long amount) {  
        long aux = this.balance;  
        aux = aux + amount;  
        this.balance = aux;  
    }
```

```
    void withdraw (long amount)
```

```
    {  
        long aux = this.
```

```
        if (aux >= amount
```

```
            aux = aux - am
```

```
            this.balance =
```

```
        }  
    }
```

```
}
```

```
Account acc = new .
```

Проблем:

- методите не са неделими
- времеви интервал може да свърши по средата на метода

Клиент:

`acc.deposit(200)`

Пълномощно:

t1	t2	Acc
acc.deposit(200) (<i>aux</i>)	acc.withdraw(200) (<i>aux</i>)	1000
long aux = balance; (<i>1000</i>) aux = aux + amount; (<i>1200</i>)	long aux = balance; (<i>1000</i>) aux = aux - amount; (<i>800</i>)	1200
balance = aux;	balance = aux;	800

СИНХРОНИЗИРАЩИ МЕТОДИ

```
class Account {  
    private long balance;  
  
    synchronized void deposit (long amount) {  
        long aux = this.balance;  
        aux = aux + amount;  
        this.balance = aux;  
    }  
  
    synchronized void withdraw (long amount) {  
        long aux = this.balance;  
        if (aux >= amount) {  
            aux = aux - amount;  
            this.balance = aux;  
        }  
    }  
}
```

Синронизиращи методи:

Когато един Thread изпълнява синхронизиращ метод, тогава той получава Lock върху обекта:

- друг Thread няма достъп до обекта
- синхронизиращият метод завършва напълно

API-CLASS OBJECT: МЕТОДИ ЗА THREADS

java.lang

Class Object

java.lang.Object

```
public class Object
```

Class Object is the root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class.

Since:

JDK1.0

See Also:

[Class](#)

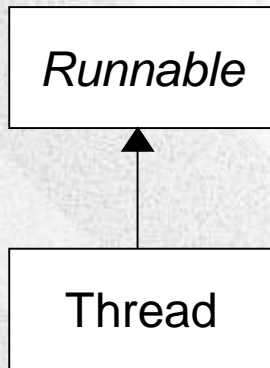
Constructor Summary

Object ()	
---------------------------	--

Method Summary

void	notify () Wakes up a single thread that is waiting on this object's monitor.
void	notifyAll () Wakes up all threads that are waiting on this object's monitor.
void	wait () Causes current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.
void	wait (long timeout) Causes current thread to wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
void	wait (long timeout, int nanos) Causes current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

API-ИЗВЛЕЧЕНИЕ: RUNNABLE



java.lang

Interface Runnable

All Known Implementing Classes:

[AsyncBoxView.ChildState](#), [FutureTask](#), [RenderableImageProducer](#), [Thread](#), [TimerTask](#)

```
public interface Runnable
```

The `Runnable` interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called `run`.

This interface is designed to provide a common protocol for objects that wish to execute code while they are active. For example, `Runnable` is implemented by class `Thread`. Being active simply means that a thread has been started and has not yet been stopped.

In addition, `Runnable` provides the means for a class to be active while not subclassing `Thread`. A class that implements `Runnable` can run without subclassing `Thread` by instantiating a `Thread` instance and passing itself in as the target. In most cases, the `Runnable` interface should be used if you are only planning to override the `run()` method and no other `Thread` methods. This is important because classes should not be subclassed unless the programmer intends on modifying or enhancing the fundamental behavior of the class.

Since:

JDK1.0

See Also:

[Thread](#)

Method Summary

void	run()
------	-----------------------

	When an object implementing interface <code>Runnable</code> is used to create a thread, starting the thread causes the object's <code>run</code> method to be called in that separately executing thread.
--	---

БЛАГОДАРЯ ЗА ВНИМАНИЕТО!

КРАЙ “НИШКИ”

