

Threads

9

Conceitos
Sincronismo
Temporizadores

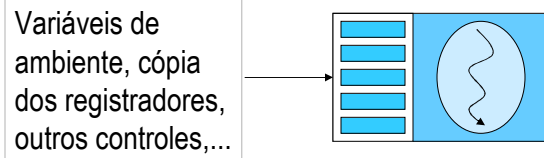
Frederico Costa Guedes Pereira © 2004
fredguedespereira@yahoo.com.br

Conceitos

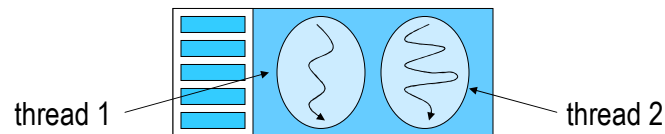
- Thread ou *light weight process*
 - Definição: linha de execução dentro de um processo
- Uma *thread* **não é** um processo!
- Um processo comum possui normalmente uma única linha de execução (*thread*)
- *Threads* de um mesmo processo compartilham informações pertencentes ao processo que as criou.

Conceitos

■ Processo *single threaded*:



■ Processo *multi-threaded*:



3

Conceitos

■ Exemplos de aplicações *multi-threaded*:

- Browser Internet:
 - *threads* para buscar os arquivos no host
- Interpretador Java:
 - *thread* do coletor de lixo
- Interfaces gráficas:
 - *thread* do detector de eventos da GUI
- Servidor web:
 - *threads* para atendimento a vários clientes
- Jogos, etc.

4

Conceitos

- Vantagens dos *threads* sobre os processos:
 - Menos sobrecarga para criar e destruir *threads*
 - A comunicação entre *threads* é mais fácil e rápida
 - *Threads* podem compartilhar dados pertencentes ao processo originário
- O sistema operacional é quem oferece suporte à existência de *threads* numa linguagem de programação

5

Exemplo simples

```
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 {  
                Thread.sleep((long) (Math.random() * 1000));  
            } catch (InterruptedException e) {}  
        }  
        System.out.println("DONE! " + getName());  
    }  
}
```

*Retirado do livro The Java Tutorial

6

Exemplo simples

```
public class TwoThreadsDemo {  
    public static void main  
    (String[] args) {  
        new SimpleThread("Thread  
1").start();  
        new SimpleThread("Thread  
2").start();  
    }  
}
```

```
sequenceDiagram  
    participant T1 as Thread 1  
    participant T2 as Thread 2  
    participant TD as TwoThreadsDemo  
    TD->>T1: start()  
    activate T1  
    TD->>T2: start()  
    activate T2  
    T2->>T2: sleep()  
    deactivate T2  
    T1-->>TD: end  
    deactivate T1
```

*Retirado do livro The Java Tutorial

7

Conceitos

- Como criar múltiplas *threads* em Java?
 - Especializar a classe `java.lang.Thread`
 - Implementar a interface `java.lang.Runnable`
- O código concorrente deve ser colocado no método `run()` da classe
- Para disparar uma nova *thread*, instancie a classe e envie `start()` ao objeto recém-criado
- Não chame `run()` diretamente!

8

Conceitos

■ API de Java:

- **Thread()**: cria uma nova linha de execução.
- void **run()**: método que contém a linha de execução.
- void **start()**: prepara a linha de execução e chama run() internamente.
- static void **sleep(long ms)**: põe a linha de execução corrente em suspensão por ms milisegundos. Este método lança uma exceção **InterruptedException** para indicar que o tempo indicado expirou.

9

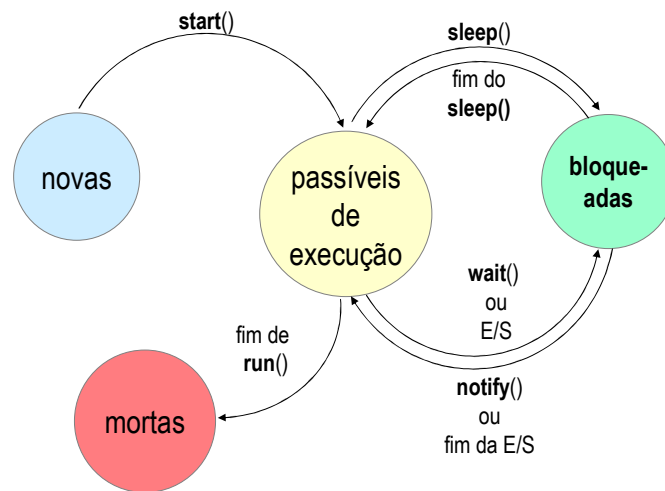
Estados de uma thread

■ Linhas de execução podem ser:

- **Novas**: o objeto foi instanciado (new) mas não iniciado (start)
- **Passíveis de execução**: estão prontas para serem escalonadas pelo sistemas operacional ou estão executando no momento (não há como diferenciar)
- **Bloqueadas**: aguardando o término de uma operação de E/S; executando um sleep(); executou um wait() sobre um objeto; bloqueada por tentar usar um objeto bloqueado.
- **Mortas**: estado apenas representativo

10

Estados de uma thread

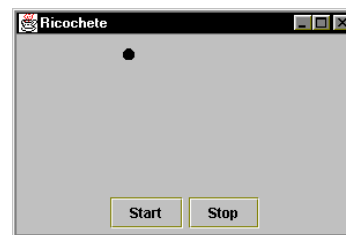


11

Exemplo mais completo

■ O código:

```
import java.awt.*;  
import java.awt.event.*;  
import javax.swing.*;  
  
public class BounceThread {  
    public static void main(String[] args) {  
        JFrame frame = new BounceThreadFrame();  
        frame.show();  
    }  
}
```



12

Exemplo mais completo

```
public class BounceThreadFrame extends JFrame {
    private Ball b;

    public BounceThreadFrame() {
        Container contentPane = getContentPane();
        final JPanel canvas = new JPanel();
        contentPane.add(canvas, "Center");
        JPanel p = new JPanel();

        addButton(p, "Start",
            new ActionListener() {
                public void actionPerformed(ActionEvent evt) {
                    b = new Ball(canvas);
                    b.start();
                }
            }
        );
    }
}
```

(continua) 13

Exemplo mais completo

```
        addButton(p, "Stop",
            new ActionListener() {
                public void actionPerformed(ActionEvent evt) {
                    b.stopIt();
                }
            }
        );
        contentPane.add(p, "South");
    } //fim do construtor

    public void addButton(Container c, String title,
        ActionListener a) {
        JButton b = new JButton(title);
        c.add(b);
        b.addActionListener(a);
    }
} //fim da classe BounceThreadFrame
```

14

Exemplo mais completo

```
class Ball extends Thread {  
    private JPanel box;           //onde desenho a bola  
    private boolean stop = false; //para o thread  
  
    public Ball(JPanel b) {  
        box = b;  
    }  
  
    public void draw() {  
        desenha uma oval no JPanel  
    }  
  
    public void move() {  
        desenha uma oval da cor do background;  
        calcula novas coordenadas da bola;  
        desenha uma oval com estas coordenadas;  
    }  
}
```

15

Exemplo mais completo

```
(pode lançar)  
    public void run() {  
        try {  
            draw();  
            while (!stop) {  
                move();  
                sleep(5);  
                System.out.println(this + ": rodando...");  
            }  
        } catch (InterruptedException e) {  
        } finally {  
            System.out.println("Fim de " + this);  
            box.repaint();  
        }  
    }  
    public void stopIt() {  
        stop = true;  
    }  
} //fim da classe Ball
```

16

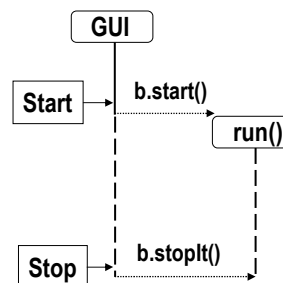
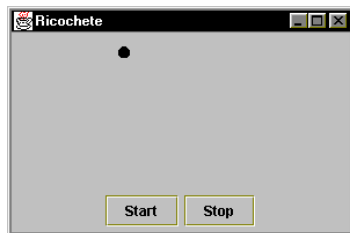
Sobre o exemplo...

- Quando um objeto **Ball** é instanciado (recebendo um **JPanel**) e recebe **start()**, uma nova thread é criada e o método **run()** começa a executar.
- O método **run()** vai desenhar uma bola no **JPanel** recebido até que a variável **stop** seja **true**.
- Esta variável será **true** quando o objeto **Ball** receber o método **stopIt()** de uma outra thread.

17

Sobre o exemplo...

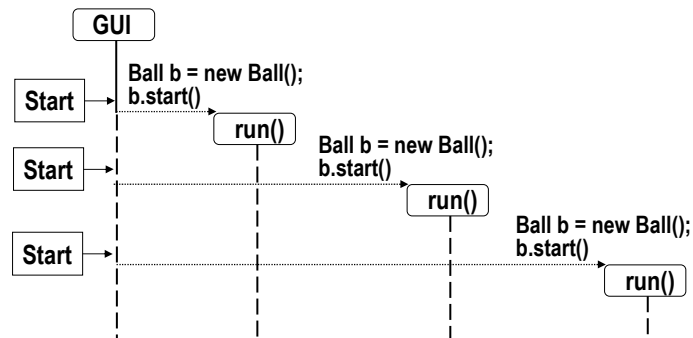
- O usuário pressiona Start e depois Stop:



18

Sobre o exemplo...

- O usuário pressiona várias vezes Start:



19

Implementando **Runnable**

- Quando uma classe já possui uma hierarquia, a herança a partir de **Thread** torna-se impossível
- Nestes casos, a opção é a classe implementar a interface **Runnable** em `java.lang`
- Runnable define o método:
 - `public void run()`
- Um Runnable pode ser passado como parâmetro de um construtor de Thread
 - Deve-se criar uma thread para executar um Runnable

20

Exemplo

```
public class Clock extends Applet
    implements Runnable {
    private Thread clockThread = null;
    private boolean stop = false;

    public void start() {
        if (clockThread == null) {
            clockThread =
                new Thread(this, "Clock");
            clockThread.start();
        }
    }
}
```

21

Exemplo

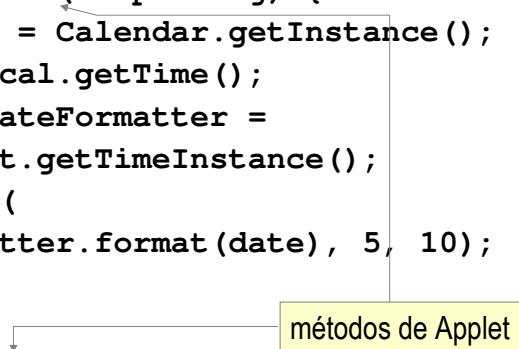
```
public void run() {
    while (!stop) {
        repaint();
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {}
    }
}
```

controle para interromper a thread

22

Exemplo

```
public void paint(Graphics g) {  
    Calendar cal = Calendar.getInstance();  
    Date date = cal.getTime();  
    DateFormat dateFormatter =  
        DateFormat.getTimeInstance();  
    g.drawString(  
        dateFormatter.format(date), 5, 10);  
}  
  
public void stop() {  
    stop = true;  
}  
} //fim da classe Clock
```



métodos de Applet

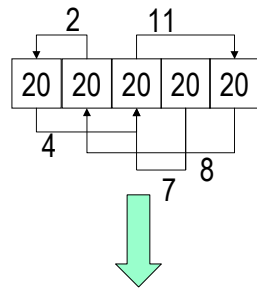
23

Sincronismo

- Threads concorrentes podem compartilhar dados e objetos
- O acesso a estes dados deve ser organizado para que os mesmos não fiquem num estado inconsistente
- Duas formas de se obter o sincronismo:
 - bloqueando o objeto compartilhado (synchronized)
 - uso de wait() e notify()

24

Sincronismo



18 26 20 13 23

```
class Banco {  
    int[] c;  
    transfere(int d, int p, int v) {  
        if(c[d] < v) return;  
        c[p] += v;  
        c[d] -= v;  
    }  
}
```

```
class TransfereThread {  
    Banco b;  
    public void run() {  
        b.transfere(de, para, valor);  
    }  
}
```

25

Sincronismo

- O método transfere não é atômico:

```
void transfere(int d, int p, int v) {  
    if(c[d] < v) return;  
    c[p] += v;  
    c[d] -= v;  
}
```

escalonamento!

- Solução:

```
void synchronized transfere(int d, int p, int v){  
    if(c[d] < v) return;  
    c[p] += v;  
    c[d] -= v;  
}
```

ATÔMICO

26

Sincronismo

- O modificador **synchronized** bloqueia o objeto para a thread que primeiro entrou no método.
- Nenhuma outra thread poderá solicitar um método synchronized sobre o objeto.
- Ao término da execução do método, o objeto é desbloqueado.
- O synchronized define monitores na linguagem Java.

27

Sincronismo

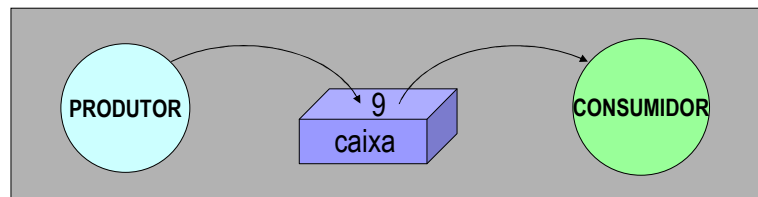
- O modificador **synchronized** pode ser usado de forma mais pontual, evitando o bloqueio completo do objeto.

```
void atualiza() {  
    int[] dados;  
    ...  
    synchronized(dados) {  
        modifica dados  
    }  
    ...  
}
```

28

Produtor/Consumidor

- A classe Produtor gera inteiros entre 0 e 9 e armazena-os na classe Cesta
- Em seguida, dorme por um intervalo aleatório entre 0 e 100
- A classe Consumidor retira inteiros da Cesta



29

Exemplo

```
public class Produtor extends Thread {
    private Caixa caixa;
    private int num;

    public Produtor(Caixa c, int n) {
        this.caixa = c; this.num = n;
    }

    public void run() {
        for (int i = 0; i < 10; i++) {
            caixa.put(i);
            System.out.println("P #" + num + " pôs: " + i);
            try {
                sleep((int) (Math.random() * 100));
            } catch (InterruptedException e) { }
        }
    }
}
```

30

Exemplo

```
public class Consumidor extends Thread {
    private Caixa caixa;
    private int num;

    public Consumidor(Caixa c, int n) {
        caixa = c;    this.num = n;
    }

    public void run() {
        int value = 0;
        for (int i = 0; i < 10; i++) {
            value = caixa.get();
            System.out.println("Consumidor #" +
                               this.num + " pegou: " + value);
        }
    }
}
```

31

Exemplo

```
public class Caixa {
    private int conteudo;
    private boolean disponivel = false;

    public synchronized int get() {
        ...
    }

    public synchronized void put(int value) {
        ...
    }
}
```

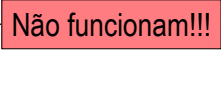
32

Exemplo

```
public class Caixa {
    private int conteudo;
    private boolean disponivel = false;

    public synchronized int get() {
        if (disponivel == true) {
            disponivel = false;
            return conteudo;
        }
    }

    public synchronized void put(int valor) {
        if (disponivel == false) {
            disponivel = true;
            conteudo = valor;
        }
    }
}
```



Não funcionam!!!

33

Exemplo

```
public class Caixa {
    private int conteudo;
    private boolean disponivel = false;

    public synchronized int get() {
        while (disponivel == false) {
            try {
                wait();
            } catch (InterruptedException e) {}
        }
        disponivel = false;
        int temp = conteudo;
        notifyAll();
        return temp;
    }
}
```

34

Exemplo

```
public synchronized void put(int valor) {  
    while (disponivel == true) {  
        try {  
            wait();  
        } catch (InterruptedException e) { }  
    }  
    conteudo = valor;  
    disponivel = true;  
    notifyAll();  
}
```

35

Temporizadores

- Java possui as classes **Timer** e **TimerTask** no pacote **java.util** para fazer temporização.
- Internamente estas classes utilizam Threads, livrando o programador da tarefa de conhecer os detalhes de implementação de uma thread.
- A classe **TimerTask** é abstrata e deve ser especializada para conter a tarefa a ser escalonada.

36

Temporizadores

- Classe Timer:
 - `public Timer()`
 - `schedule(TimerTask task, Date time)`
 - `schedule(TimerTask task, long delay)`
 - `void cancel()`
- Classe TimerTask:
 - `abstract void run()`

37

Temporizadores

```
public class Reminder {
    Timer timer;

    public Reminder(int seconds) {
        timer = new Timer();
        timer.schedule(new RemindTask(),
            seconds*1000);
    }

    class RemindTask extends TimerTask {
        public void run() {
            System.out.println("Acabou!");
            timer.cancel();
        }
    }
}
```

38

Temporizadores

```
public static void main(String args[]) {  
    System.out.println("Escalonador...");  
    new Reminder(5);  
    System.out.println("Tarefa  
                        programada");  
}  
}
```