**ồng**

**Bài 12**

**Đa lu**

**Mục tiêu**

Giới thiệu luồng – Thread ▪

dụng luồng Tạo và sử

▪

Quản lý luồng ▪

Luồng chính – deamon thread ▪

Đa luồng ▪

Khóa và bất đồng bộ ▪

Deadlock và khắc phục ▪

**Multithreading**

**Giới thiệu**

được th

**trình** Một **process** là một **chương** ▪

Mỗi **process** có nguồn tài nguyên run-timemình chẳng hạn như **dữ liệu**, **biến**, **không nhớ**. ▪

là đơn vị cơ bản mà hệ điều hành

thời gian xử lý.

**Thread**

▪

là thực thể trong một **process** có

hoạch định thực hiện.

**Thread**

▪

duy

Một **process** được bắt đầu với **thread** ▪

**Multithreading**

**chính**.

là **thread**

thời gian thực

là đơn vị nhỏ nhất của mã thực thi

vụ, công việc cụ

**thời**,

có thể thực hiện **đồng**

**lúc cùng vụ**

**Giới thiệu**

có thể chạy **độc lập Thread** ▪

**e**) **tim**

**Thread** ▪

dụng mà thực hiện nhiệm

Nhiều **Thread** ▪

kiện thực hiện một số **nhiệm**

dụng.

**Multithreading**

**Giới thiệu**

Ứng dụng thread trong lập trình là:

Phát nhạc và hiển thị hình ảnh cùng lúc. ▪

Hiển thị nhiều hình ảnh trên màn hình. ▪

Hiển thị di chuyển mẫu văn bản hoặc hìnhmàn hình. ▪

**Multithreading**

**dụng Tạo và sử**

Có 2 cách để tạo Thread.

Tạo lớp kế thừa lớp Thread.

Thực thị interface Runnable.

**Multithreading**

**Kế thừa class Thead**

Các bước tạo Thread từ class.

Thread

**Tạo class kế thừa từ class**

**Ghi đè phương thức** run()

**Khởi chạy Thread**

**Multithreading**

**Kế thừa class Thead**

Code demo: tạo class **MyThread**

Code Snippet

//Extending

Thread

extends

MyThreadclass

class

definition //class

. .

.

**Multithreading**

{

}

**Kế thừa class Thead**

Code demo: Override phương thức **run**()

c

Thread

//Extending Thread

definition

method  run()

**Multithreading**

extends

run()

the

implementation

❑

Code Snippet

MyThreadclass

class

//

{

voidpublic

//overriding

//

{

}

.

.

.

}

**Kế thừa class Thead**

Code demo: Khởi chạy thread

Code Snippet

TestThread  class

public

args[])  main(String

void

static

public

thread

//creating MyThread(); t=new

MyThread

thread  the

//Starting

t.start();

**Multithreading**

{

. .

.

{

}

}

**Descriptio**

**Contructors của Thread**

Default constructor

Tạo thread với tham số là mộTạo thread với tham số là mộcủa Thread (chuỗi)  String

Tạo thread với tham số là tênTạo thread với tham số là mộvà một Runnable

**Multithreading**

**Constructor**

objRun)

objRun,

threadName)

group,

Thread()

Thread(Runnable

Thread(Runnable

threadName)

Thread(String

Thread(ThreadGroup

objRun)Runnable

**Descripti**

**Phương thức của Thread**

**Multithreading**

Trả về số luồng đang active tcủa chương trình.

Trả về tham chiếu đến đối tthực hiện.

ThreadGroup mà

thuộc về

Trả về

tra thread hiện tại có bKiểm

tra thread còn sống (hKiểm

tra thread đã bị ngắt cKiểm

Đợi thread hiện tại dừng

Đặt tên cho thread.

**Method**

activeCount()  int

static

currentThread()  Thread

static

getThreadGroup() ThreadGroup

interrupted()  boolean

static

isAlive() boolean

boolean isInterrupted()

join() void

name)  setName(String

void

**Demo**

\* Creating threads using Thread class and using methods of the class

\* NamedThread is created as a subclass of the class Thread

**Multithreading**

public class NamedThread extends Thread {

/\* This will store name of the thread \*/

Code demo:

String name;

package demo;

\* This method of Thread class is overridden to specify the action

\* that will be done when the thread begins execution

/\*\*

\*/

/\*\*

\*/

/\*\*

\*/

**Multithreading**

//Display the name of the currently running thread

name = Thread.currentThread().getName();

System.out.println(Thread.activeCount());

System.out.println(“Marimba”);

// Will store the number of threads

//Display the number of threads

System.out.println(name);

if (name.equals (“Thread1”))

System.out.println(“Jini”);

while(count<=3) {

❑public void run() {

**Demo**

int count = 0;

count++;

else

}

}

**Demo**

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

NamedThread(); NamedThread objNamedThread = new

objNamedThread.setName(“Thread1”);

//Display the status of the thread, whether alive or not

System.out.println(Thread.currentThread().isAlive());

System.out.println(objNamedThread.isAlive());

/\*invokes the start method which in turn will call

run and begin thread execution

\*/

objNamedThread.start();

System.out.println(Thread.currentThread().isAlive());

System.out.println(objNamedThread.isAlive());

}

}

**Multithreading**

**Interface Runnable**

từ interface **Runnable**: Tạo **Thread**

**Interface Runnable** được thiết kế để cuntập các quy tắc cho các đối tượng có nhu chiện mã trong một thread. ▪

Java không cho phép **đa kế thừa** nên cáchnày là một giải pháp khác.▪

**Multithreading**

**Interface Runnable**

từ interface **Runnable**: Tạo **Thread**

**Thực thi interface** Runnable

**Thực thi phương thức** run()

**Khởi chạy thread**

**Multithreading**

**Interface Runnable**

Code demo: tạo class thực thi interface

Code Snippet

Runnable

implements that

class

a

Declaring//

Runnable

implements

MyRunnable

class

{

.

.

.

}

**Multithreading**

Code demo: thực thi phương thức **run**()

Runnable

Runnable

Overri

**Interface Runnable**

Code Snippet

implements that class a

Declaring

implements MyRunnable class

//

run() void

public

{

implementation

.

.

**Multithreading**

//

interface

{

Run()

.

//

}

}

args[])

MyRunnable();

thread

main(String

Thread(r);

a

**Multithreading**

**Interface Runnable**

Code demo: khởi chạy thread

Code Snippet

ThreadTest class

void staticpublic

r=new

MyRunnable

{

thObj=newThread

//StartingthObj.start();

}

{

}

**Demo code**

**Multithreading**

/\*

\*class and using methods of the class

\*Creating threads using Thread

\*/

❑

package test;

/\*\*

\* NamedThread is created so as to implement the interface Runnable

\*/

class NamedThread implements Runnable {

/\* this will store name of the thread \*/

String name;

/\*\*

\*

\* This method of Runnable is implemented to specify the action

when the thread begins execution.

\*/

public void run() {

int count = 0; //will store the number of threads

while(count < 3){

**Demo code**

System.out.

name = Thread.currentThread().getName();

count++;

❑

}

}

}

public class Main {

public static void main(String args[])

{

NamedThread() NamedThread objNewThread= new

objThread.sThread objThread = new Thread(objNewThread);

}

}

**Multithreading**

Blocked

State

notify(), notifyAll()

**Multithreading**

Waiting

wait()

State

Terminat

State

**Vòng đời**

start()

Thread

New

ed

**Vòng đời**

**Multithreading**

Giải thích:

**New:** Một thread mới bắt đầu vòng đời củatrạng thái new. Nó tồn tại trong trạng tháichương trình bắt đầu thread này. Nó cũngnhư là một thread mới sinh.

**Runnable:** Sau khi một thread mới sinh rađầu, thread trở thành runnable. Một threadtrạng thái này được xem như đang thực hicủa nó.

waiting trong khi thread đợi cho thread kháhiện một tác vụ. Một thread chuyển về trạrunnable chỉ khi thread khác ra hiệu cho thđợi để tiếp tục thực thi.

**aiting:** Đôi khi, một thread quá độ qua t

**W**

▪

▪

▪

**Vòng đời**

**ed waiting:** Một thread trong trạng th**Tim**

▪

runnable có thể đi vào trạng thái timed waimột khoảng thời gian nào đó. Một thread tthái này chuyển về trạng thái runnable khithời gian đó kết thúc hoặc khi sự kiện nó đxuất hiện.

**inated:** Một thread trong trạng thái r**Term**

▪

có thể đi vào trạng thái terminated khi nótác vụ của nó hoặc nó chấm dứt.

**Multithreading**

**Phương thức class Thread**

Một số phương thức quan trọng trong lớp Thre⮚ getName()

start() ⮚

run() ⮚

sleep() ⮚

interrupt()

⮚

**Multithreading**

**Quản lý luồng**

là đơn vị có khả năng tự hoạt động

chương trình.

**Thread**

▪

Trong một chương trình đơn lẻ, các threadhoạt động độc lập. ▪

Đôi khi, các luồng lại dùng chung tài nguyê▪

Và, trong vài tình huống luồng đang chạy cchặn để tài nguyên có thể sử dụng chỉ 1 lucùng một thời điểm. ▪

soát nội bộ, quản lý luồng

nhất để chương trình thông suốt.

process

Do đó cần kiểm

▪

T7 T6 T5 T4 T3 T2 T1

**Multithreading**

**Độ ưu tiên**

Trong tình huống tài nguyên hoặc xử lý cóchấp giữa nhiều luồng thì cần xác định độ ▪

quan trọng của thr

Độ ưu tiên thể hiện tầm ▪

khác nhau.

**Multithreading**

Độ ưu tiên là số nguyên dao động giữa:

và **MAX\_PRIORITY**.

ưu ti

Luồng có chỉ số ưu tiên cao được CPU

Thread.MAX\_PRIORITY: Hằng số 10,

Thread.NORM\_PRIORITY: Hằng số 5,bình.

Thread.MIN\_PRIORITY: Hằng số 1. th

**Multithreading**

**Độ ưu tiên**

▪

**MIN\_PRIORITY**

▪

hơn.

Các mức ưu tiên: ▪

1.

2.

3.

**Deamon Thread**

Một deamon thread chạy liên tục để thực hdịch vụ mà không cần bất kỳ kết nối với tochương trình. ▪

của deamon thread:

Các đặc điểm ▪

• Hoạt động nền, cung cấp dịch vụ cho các thread khác.

**1**

• Phụ thuộc vào luồng người dùng.

**2**

dừng một lần các luồng chết và chỉ deamon thr**Multithreading**• Máy ảo JVM

3

fro

derived

cla the of

\*/

thread

{

to

overridden

begi thread

**Đa luồng**

Code demo: thực thi đa luồng

class a

using threads multiple Creating

class

test;

subclass a

as

created is

MultipleThreads

Thread extends MultipleThreads class

the of name

the store to

Variable

name; String

is

class Thread of method This

the when done

be

will thataction

\*/

**Multithreading**

/\*\* \*

package

\*/

/\*\* \*

public

\*/

/\*

/\*\* \*

Thread.currentThread().getName()

System.out.println(name);

Thread.sleep(500);

e)  InterruptedException

break;

MultipleThr the for

point

**Multithreading**

{

run()

{

=

{

}

catch(

{

}

}

entry the

**Đa luồng**

void

while(true)

name

try

}

is

This

\*/

public

/\*\* \*

”

MultipleThreads();

{

args[])

MultipleThreads();

running: threads

**Multithreading**

main(String void

new

=

t1MultipleThreads

new

=

t2MultipleThreads

t1.setName(“Thread2”);

t2.setName(“Thread3”);

t1.start();

ofSystem.out.println(“Number

activeCount());

**Đa luồng**

staticpublic

t2.start();

}

}

**Phương thức của Thread**

tra luồng có đang hoạt động k

args)  []

System.out.println(“The

ThreadDemo();

**Multithreading**

**isAlive**(): kiểm

Code Snippet

main(String

void

static

public

.

.

.

new

=

Obj

ThreadDemo

{

Thread(Obj); new

=

t

Thread

t.isAlive());

+

:”

alive

}

. .

.

**Phương thức của Thread**

**join**(): đợi luồng hiện tại dừng

Code Snippet

for

waiting and main the in

am

System.out.println(“I

object  Thread

e)  InterruptedException

interrupted”);  is

thread

System.out.println(“Main

**Multithreading**

finish”);

a

is

objTh.join();

try

{

to

objTh//

catch(

}

{

}

. .

.

**Điều kiện tranh chấp**

Trong chương trình đa luồng, một số luồngđồng thời cố gắng cập nhật nguồn tài nguyví dụ như tập tin. ▪

Điều này khiến tài nguyên xung đột. ▪

Do đó cần xác định điều kiện tranh chấp chluồng. ▪

• Hai hay nhiều luồng cùng chia sẻ tài nguyên, dữ liệu.

1

• Hai hay nhiều luồng thực hiện đọc/ghi dữ liệu đồng tnguyên dữ liệu.

2

❑

**Multithreading**

**Điều kiện tranh chấp**

Khối Synchronized: áp dụng cho object

block

synchronized(object)

synchronized  be

to

statements

method(...)

method

method  of

**Multithreading**

//synchronized

//

{

//synchronized

}

synchronized

body

//

{

}

**Điều kiện tranh chấp**

Khối Synchronized: áp dụng cho phương thức

value)

deposit(float

void

0.0;

value; +

balance

**Multithreading**

Code Snippet

Account

class

.

.

=

balancefloat

synchronizedpublic

=

balance

{

}

. .

.

{

}

.

**Điều kiện tranh chấp**

Demo Synchronized phương thức

methods.  synchronized

Demonstrating

test;

saf

thread

the

use

to

synchronized

is

method

{

{

num)

display(int

void

synchronized

num); +

System.out.print(“”

System.out.println(“Interrupted”);

{

e)

catch(InterruptedException

Thread.sleep(1000);

done”); System.out.println(“

**Multithreading**

/\*\* \*

package

\*/

One

class

This//

num++;

{

try

}

}

}

}

**Điều kiện tranh chấp**

class Two extends Thread {

int number;

One objOne;

public Two(One one\_num, int num) {

objOne = one\_num;

number = num;

}

public void run() {

// Invoke the synchronized method

objOne.display(number);

}

}

class SynchMethod {

public static void main(String args[]) {

One objOne = new One();

int digit = 10;

**Multithreading**

**Điều kiện tranh chấp**

khác biệt

3 thread khởi chạy đồng thời, điểm

yêu cầu syschronized là giữa các luồng sẽ chờ// Create three thread objects

Two(objOne);

Two objSynch1 = new

Two(objOne); Two objSynch2 = new

Two(objOne); Two objSynch3 = new

objSynch1.start();

objSynch2.start();

objSynch3.start();

}

}

**Multithreading**

**Phương thức của Thread**

c

**wait**(): luồng hiện tại rơi vào trạng thái tạm

Code Snippet

takeup()  void

synchronized

{

(!available)

the o for

waiting is

System.out.println(“Philosopher

chopstick”);

wait();

e)  InterruptedException

catch(

false;

**Multithreading**

public

.

.

while

{

try

}

{

}

=

available

. .

.

{

}

}

.

**Phương thức của Thread**

**notify**(): thông báo cho các luồng khác đang Code Snippet

putdown()  void

synchronized

true; =

**Multithreading**

public

.

.

.

available

{

notify();

}

. .

.

**Deadlock**

**bế tắc**: mô tả tình huống mà **Deadlock –**

▪

nhiều luồng bị chặn mãi mãi do phải chờ đnguyên được giải phóng.

Đôi khi tình huống xảy ra khi 2 luồng bị khnguyên của mình, chờ đợi ổ khóa tương ứnđổi với nhau. ▪

**Multithreading**

interface.  Runnable

Runnable  implements

args[])

DeadlockDemo();

DeadlockDemo();

(objDead1);

(objDead2);

**Multithreading**

the

main(String

new =

new =

Thread

Thread

objDead2;

objDead1;

**Deadlock**

Demo:

Deadlock.  Demonstrating

test; package

implements DeadlockDemo

DeadlockDemo classpublic

void staticpublic

objDead1

DeadlockDemo

{

objDead2DeadlockDemo

new

=

objTh1Thread

new

=

objTh2Thread

=

objDead1.grabIt

=

objDead2.grabIt

/\*\* \*

\*/

/\*\* \*

\*/

{

System.out.println(“Started”);

occurred”);

{

e)

objTh1.start();

objTh2.start();

objTh1.join();

objTh2.join();

InterruptedExceptioncatch(

System.out.println(“error

System.exit(0);

{

run()

void

grabIt;

synchronized

Thread.sleep(500);

**Multithreading**

**Deadlock**

{

try

}

}

DeadlockDemo

}

public

{

try

{

occur

occurred”);

method”);

**Deadlock**

e) InterruptedException catch(}

System.out.println(“error

grabIt.syncIt();

}

{

syncIt()

void

synchronized

System.out.println(“Sync”);

Thread.sleep(500);

{

System.out.println(“error

{

e)

catch(InterruptedException

syncIt() theSystem.out.println(“In

class

**Multithreading**

public

}

try

}

}

end

}

}//

**Deadlock**

Tránh deadlock:

• Tránh nhận nhiều khóa tại một thời điểm 1

bảo trong chương trình có các ổ khóa theo thứ tự

được xác định

• Đảm

2

**Multithreading**

nhiều **thread**.

giúp chương trình có thể thực hiệ

Có 2 cách tạo luồng là: kế thừa class **Thr**thực thi **interface Runnable**.

Độ ưu tiên xác định độ quan trọng giữa c

cung cấp dịch vụ cho cá

Điều kiện tranh chấp giải quyết bởi khối

**Deadlock** là tình huống nhiều luồng cùnkhóa chung được giải phóng.

**Multithreading**

**tắt bài học**

**Process** có thể bao gồm

tác vụ đồng thời.

**Thread**

**thread on**

**synchronized**.

**Tóm**

**Deam**

khác.

✔

✔

✔

✔

✔

✔

✔