Примеры обработки исключений

**package** \_java.\_se.\_07.\_exception;

**public class** ThreadUncaughtExceptionDemo {

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

Thread t = **new** Thread(**new** SimpleThread());

t.setUncaughtExceptionHandler(**new** Thread.UncaughtExceptionHandler() {

**publicvoid** uncaughtException(Thread t, Throwable e) {

System.*out*.println(t + " throws exception: " + e);

}

});

t.start();

}

}

**class** SimpleThread **implements** Runnable {

**publicvoid** run() {

**thrownew** RuntimeException("It is a greate exception.");

}

}

**package** \_java.\_se.\_07.\_exception;

**publicclass** ThreadDefaultUncaughtExceptionDemo {

**publicstaticvoid** main(String[] args) {

Thread.*setDefaultUncaughtExceptionHandler*(**new** Thread.UncaughtExceptionHandler() {

**publicvoid** uncaughtException(Thread t, Throwable e) {

System.*out*.println(t + " (default handler)throws exception: " + e);

}

});

Thread t1 = **new** Thread(**new** MyThread());

Thread t2 = **new** Thread(**new** MyThread());

t2.setUncaughtExceptionHandler(**new** Thread.UncaughtExceptionHandler() {

**publicvoid** uncaughtException(Thread t, Throwable e) {

System.*out*.println(t + " throws exception: " + e);

}

});

t1.start();

t2.start();

}

}

**class** MyThread **implements** Runnable {

**publicvoid** run() {

**thrownew** RuntimeException();

}

}

**package** \_java.\_se.\_07.\_exception;

**publicclass** ThreadGroupUncaughtExceptionDemo {

**publicstaticvoid** main(String[] args) {

NewThreadGroup g = **new** NewThreadGroup("one");

ThreadD t1 = **new** ThreadD("1", g);

ThreadD t2 = **new** ThreadD("2", g);

ThreadD t3 = **new** ThreadD("3", g);

t3.setUncaughtExceptionHandler(**new** Thread.UncaughtExceptionHandler() {

**publicvoid** uncaughtException(Thread t, Throwable e) {

System.*out*.println(t + " throws exception: " + e);

}

});

t1.start();

t2.start();

t3.start();

}

}

**class** NewThreadGroup **extends** ThreadGroup {

NewThreadGroup(String n) {

**super**(n);

}

NewThreadGroup(ThreadGroup parent, String n) {

**super**(parent, n);

}

**publicvoid** uncaughtException(Thread t, Throwable e) {

System.*out*.println(t + " has unhandled exception:" + e);

}

}

**class** ThreadD **extends** Thread {

**public** ThreadD(String threadname, ThreadGroup tgOb) {

**super**(tgOb, threadname);

}

**publicvoid** run() {

**thrownew** RuntimeException("Oy, exception!!!");

}

}

Примеры синхронизации

**package** \_java.\_se.\_07.\_synchronized;

**publicclass** Account {

**privateint**balance;

**public** Account(**int** balance){

**this**.balance = balance;

}

**publicint** getBalance(){

**return**balance;

}

**publicvoid** deposit(**int** amount){

**int** x = balance + amount;

**try** {

Thread.*sleep*(15);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

balance = x;

}

**publicvoid** withdraw(**int** amount){

**int** x = balance - amount;

**try** {

Thread.*sleep*(20);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

balance = x;

}

}

**package** \_java.\_se.\_07.\_synchronized;

**publicclass** OperatorDeposit **extends** Thread {

**private** Account account;

**public** OperatorDeposit(Account account){

**this**.account = account;

}

**publicvoid** run(){

**for**(**int** i=0; i<5; i++){

account.deposit(100);

}

}

}

**package** \_java.\_se.\_07.\_synchronized;

**publicclass** OperatorWithdraw **extends** Thread {

**private** Account account;

**public** OperatorWithdraw(Account account) {

**this**.account = account;

}

**publicvoid** run() {

**for** (**int** i = 0; i < 5; i++) {

account.withdraw(50);

}

}

}

**package** \_java.\_se.\_07.\_synchronized;

**publicclass** OperationInspector {

**publicstaticvoid** main(String[] args) **throws** InterruptedException {

Account account = **new** Account(200);

OperatorDeposit opD = **new** OperatorDeposit(account);

OperatorWithdraw opW = **new** OperatorWithdraw(account);

opD.start();

opW.start();

opD.join();

opW.join();

System.*out*.println(account.getBalance());

}

}

**package** \_java.\_se.\_07.\_synchronized;

**public class** Account {

**privateint**balance;

**public** Account(**int** balance){

**this**.balance = balance;

}

**publicint** getBalance(){

**return**balance;

}

**public synchronized void** deposit(**int** amount){

**int** x = balance + amount;

**try** {

Thread.*sleep*(15);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

balance = x;

}

**public synchronized void** withdraw(**int** amount){

**int** x = balance - amount;

**try** {

Thread.*sleep*(20);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

balance = x;

}

}

**package** \_java.\_se.\_07.\_synchronized;

**publicclass** Account {

**privateint**balance;

**private** Object lock = **new** Object();

**public** Account(**int** balance){

**this**.balance = balance;

}

**publicint** getBalance(){

**return**balance;

}

**publicvoid** deposit(**int** amount){

**synchronized** (lock) {

**int** x = balance + amount;

**try** {

Thread.*sleep*(15);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

balance = x;

}

}

**publicvoid** withdraw(**int** amount){

**synchronized** (lock) {

**int** x = balance - amount;

**try** {

Thread.*sleep*(20);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

balance = x;

}

}

}

**package** \_java.\_se.\_07.\_synchronized;

**publicclass** Account {

**privateint**balance;

**public** Account(**int** balance){

**this**.balance = balance;

}

**publicint** getBalance(){

**return**balance;

}

**publicvoid** deposit(**int** amount){

**synchronized** (**this**) {

**int** x = balance + amount;

**try** {

Thread.*sleep*(15);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

balance = x;

}

}

**publicvoid** withdraw(**int** amount){

**synchronized** (**this**) {

**int** x = balance - amount;

**try** {

Thread.*sleep*(20);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

balance = x;

}

}

}

**package** \_java.\_se.\_07.\_synchronized;

**publicclass** OperatorDeposit **extends** Thread {

**private** Account account;

**public** OperatorDeposit(Account account){

**this**.account = account;

}

**publicvoid** run(){

**for**(**int** i=0; i<5; i++){

**synchronized** (account) {

account.deposit(100);

}

}

}

}

**package** \_java.\_se.\_07.\_synchronized;

**public class** OperatorWithdraw **extends** Thread {

**private** Account account;

**public** OperatorWithdraw(Account account) {

**this**.account = account;

}

**public void** run() {

**for** (**int** i = 0; i < 5; i++) {

**synchronized** (account) {

account.withdraw(50);

}

}

}

}

Еще синхронизация

**package** \_java.\_se.\_07.\_synchronized;

**class** StaticSynch{

**publicstaticsynchronizedvoid** a() **throws** InterruptedException{

System.*out*.println("Line #1 in the method a");

Thread.*sleep*(1000);

System.*out*.println("Line #2 in the method a");

}

**publicstaticsynchronizedvoid** b() **throws** InterruptedException{

System.*out*.println("Line #1 in the method b");

Thread.*sleep*(1000);

System.*out*.println("Line #2 in the method b");

}

}

**publicclass** StaticSynchnizedDemo {

**publicstaticvoid** main(String[] args){

**new** Thread(){

**publicvoid** run(){

**for**(**int** i=0; i<5; i++){

**try** {

StaticSynch.*a*();

Thread.*sleep*(20);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

}.start();

**for**(**int** i=0; i<5; i++){

**try** {

StaticSynch.*b*();

Thread.*sleep*(20);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

}

Использование wait, notify

**package** \_java.\_se.\_07.\_waitnotify;

**import** java.util.ArrayList;

**import** java.util.List;

**publicclass** SharedResource {

**private** List<Integer>list;

**public** SharedResource() {

list = **new** ArrayList<Integer>();

}

**publicvoid** setElement(Integer element) {

list.add(element);

}

**public** Integer getELement() {

**if** (list.size() > 0) {

**return**list.remove(0);

}

**returnnull**;

}

}

**package** \_java.\_se.\_07.\_waitnotify;

**import** java.util.Random;

**publicclass** UserResourceThread {

**publicstaticvoid** main(String[] args) **throws** InterruptedException {

SharedResource res = **new** SharedResource();

IntegerSetterGetter t1 = **new** IntegerSetterGetter("1", res);

IntegerSetterGetter t2 = **new** IntegerSetterGetter("2", res);

IntegerSetterGetter t3 = **new** IntegerSetterGetter("3", res);

IntegerSetterGetter t4 = **new** IntegerSetterGetter("4", res);

IntegerSetterGetter t5 = **new** IntegerSetterGetter("5", res);

t1.start();

t2.start();

t3.start();

t4.start();

t5.start();

Thread.*sleep*(100);

t1.stopThread();

t2.stopThread();

t3.stopThread();

t4.stopThread();

t5.stopThread();

t1.join();

t2.join();

t3.join();

t4.join();

t5.join();

System.*out*.println("main");

}

}

**class** IntegerSetterGetter **extends** Thread {

**private** SharedResource resource;

**privateboolean**run;

**private** Random rand = **new** Random();

**public** IntegerSetterGetter(String name, SharedResource resource) {

**super**(name);

**this**.resource = resource;

run = **true**;

}

**publicvoid** stopThread() {

run = **false**;

}

**publicvoid** run() {

**int** action;

**try** {

**while** (run) {

action = rand.nextInt(1000);

**if** (action % 2 == 0) {

getIntegersFromResource();

} **else** {

setIntegersIntoResource();

}

}

System.*out*.println("Поток " + getName() + " завершил работу.");

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

**private void** getIntegersFromResource() **throws** InterruptedException {

Integer number;

**synchronized** (resource) {

System.*out*.println("Поток " + getName()

+ " хочет извлечь число.");

number = resource.getELement();

**while** (number == **null**) {

System.*out*.println("Поток " + getName()

+ " ждет пока очередь заполнится.");

resource.wait();

System.*out*

.println("Поток " + getName() + " возобновил работу.");

number = resource.getELement();

}

System.*out*

.println("Поток " + getName() + " извлек число " + number);

}

}

**private void** setIntegersIntoResource() **throws** InterruptedException {

Integer number = rand.nextInt(500);

**synchronized** (resource) {

resource.setElement(number);

System.*out*.println("Поток " + getName() + " записал число "

+ number);

resource.notify();

}

}

}

Взаимная блокировка

**package** \_java.\_se.\_07.\_deadlock;

**public class** Account {

**private int** balance;

**public** Account(**int** balance) {

**this**.balance = balance;

}

**publicint** getBalance() {

**return** balance;

}

**publicvoid** deposit(**int** amount) {

balance = balance + amount;

}

**publicvoid** withdraw(**int** amount) {

balance = balance - amount;

}

}

**package** \_java.\_se.\_07.\_deadlock;

**public class** Operator **extends** Thread {

**private** Account account1;

**private** Account account2;

**public** Operator(Account account1, Account account2){

**this**.account1 = account1;

**this**.account2 = account2;

}

**public void** run(){

**for**(**int** i=0; i<3; i++){

operationDeposit(10);

}

}

**private void** operationDeposit(**int** depositSum){

**synchronized** (account1) {

System.*out*.println("Заблокирован первый счет.");

**synchronized** (account2) {

System.*out*.println("Заблокирован второй счет.");

account1.deposit(depositSum);

account2.withdraw(depositSum);

}

}

}

}

**package** \_java.\_se.\_07.\_deadlock;

**public class** OperatorDemo {

**publicstaticvoid** main(String[] args) {

Account acc1 = **new** Account(200);

Account acc2 = **new** Account(300);

Operator op1 = **new** Operator(acc1, acc2);

Operator op2 = **new** Operator(acc2, acc1);

op1.start();

op2.start();

}

}

Один из способов, позволяющих не попадать во взаимные блокировки, предполагает всегда получать блокировки для каждого потока в одном и том же порядке.

**private void** operationDeposit2(**int** depositSum){

**int** hashAcc1 = account1.hashCode();

**int** hashAcc2 = account2.hashCode();

Account acc1=**null**, acc2=**null**;

**if** (hashAcc1 < hashAcc2){

acc1 = account1;

acc2 = account2;

} **else**{

acc1 = account2;

acc2 = account1;

}

**synchronized** (acc1) {

System.*out*.println("Заблокирован первый счет.");

**synchronized** (acc2) {

System.*out*.println("Заблокирован второй счет.");

account1.deposit(depositSum);

account2.withdraw(depositSum);

}

}

System.*out*.println("Счета разблокированы.");

}

java.util.concurrent

Executors

**package** \_java.\_se.\_07.\_concurrent.\_executors;

**publicclass** SimpleThread **implements** Runnable{

**publicint**count = 0;

**publicvoid** run() {

**for** (**int** i = 0; i < 1000000; i++) {

count++;

}

System.*out*.println(count);

}

}

**package** \_java.\_se.\_07.\_concurrent.\_executors;

**import** java.util.concurrent.ExecutorService;

**import** java.util.concurrent.Executors;

**publicclass** Solution {

**publicstaticvoid** main(String[] args) {

ExecutorService ex = Executors.*newCachedThreadPool*();

ex.execute(**new** SimpleThread());

ex.execute(**new** SimpleThread());

ex.shutdown();

}

}

Возврат значений из задач. Интерфейс Callable

**package** \_java.\_se.\_07.\_concurrent.\_executors;

**import** java.util.concurrent.Callable;

**public class** CallableThread **implements** Callable<Integer> {

**publicint**count = 0;

**public** Integer call() {

**for** (**int** i = 0; i < 1000000; i++) {

count++;

}

**try** {

Thread.*sleep*(10000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

**return**count;

}

}

В данном примере метод call() вернет число после завершения операции.

**package** \_java.\_se.\_07.\_concurrent.\_executors;

**import** java.util.concurrent.ExecutionException;

**import** java.util.concurrent.ExecutorService;

**import** java.util.concurrent.Executors;

**import** java.util.concurrent.Future;

**public class** CollableSolution {

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

ExecutorService ex = Executors.*newCachedThreadPool*();

Future<Integer> s = ex.submit(**new** CallableThread());

Future<Integer> s1 = ex.submit(**new** CallableThread());

**try** {

System.*out*.println("а я уже здесь");

System.*out*.println(s.isDone());

System.*out*.println(s.get());

System.*out*.println(s1.get());

System.*out*.println(s.isDone());

} **catch** (InterruptedException e) {

e.printStackTrace();

} **catch** (ExecutionException e) {

e.printStackTrace();

}

}

}

http://habrahabr.ru/company/luxoft/blog/157273/