第２週　マルチスレッド

- Runnableインタフェース
- 抽象メソッド/クラス、インタフェースの説明
- マルチスレッドの同期
- 共有バッファを使ったスレッドの同期
- いくつスレッドを使ったサンプル
- 演習２、課題２

Runnableインタフェース （スレッドを生成する方法2）

Runnable interface

```
class   thread_name   extends   Thread {
    ……
   public void run(){  ……}
}
```

自分定義したスレッド（方法1）

```
class CountTenThread extends Thread {
  public void run() {
    for (int i = 0; i < 10; i++) {
      System.out.println("run:i = " + i);}}
}
```

```
public class ThreadTest {
  public static void main(String[] args) {
    CountTenThread ctt = new CountTenThread();
    ctt.start();
    for (int i = 0; i < 10; i++) {
      System.out.println("main:i = " + i);}}
}
```

クラススレッドのコンストラクタ

public Thread()

public Thread(String threadName)

public Thread(Runnable r)

public Thread(Runnable r, String threadName)

……

自分定義したRunnableクラス（方法2）

```
class CountTenR implements Runnable{
  public void run() {
    for (int i = 0; i < 10; i++) {
      System.out.println("run:i = " + i);}}
}
```

Runnableインタフェースの使い方

```
public class RunnableTest {
  public static void main(String[] args) {
    CountTenR ctr = new CountTenR();
    Thread ctt = new Thread(ctr);
    ctt.start();
    for (int i = 0; i < 10; i++) {
      System.out.println("main:i = " + i);}}
}
```

Thread(Runnable r) スレッドコンストラクタを呼び出す。

•Threadから導出したサブクラスの場合と同様にRunnableインタフェースを実装したクラスでもスレッドを制御するコードはrunメソッドの中に書きます。

•Runnableインタフェースを実装したクラスでスレッドを生成するには、Threadコンストラクタにそのクラスのオブジェクトへの参照を渡します。

Runnableインタフェース

java.lang
インタフェース Runnable

public interface **Runnable**

インスタンスを 1 つのスレッドで実行するすべてのクラスでは、Runnable インタフェースを実装する必要があります。このクラスは、引数のないメソッド run を定義しなければいけません。

メソッドの概要	
void	run() オブジェクトが実装するインタフェース Runnable を使ってスレッドを作成し、そのスレッドを開始すると、独立して実行されるスレッド内で、オブジェクトの run メソッドが呼び出されます。

メソッドの詳細

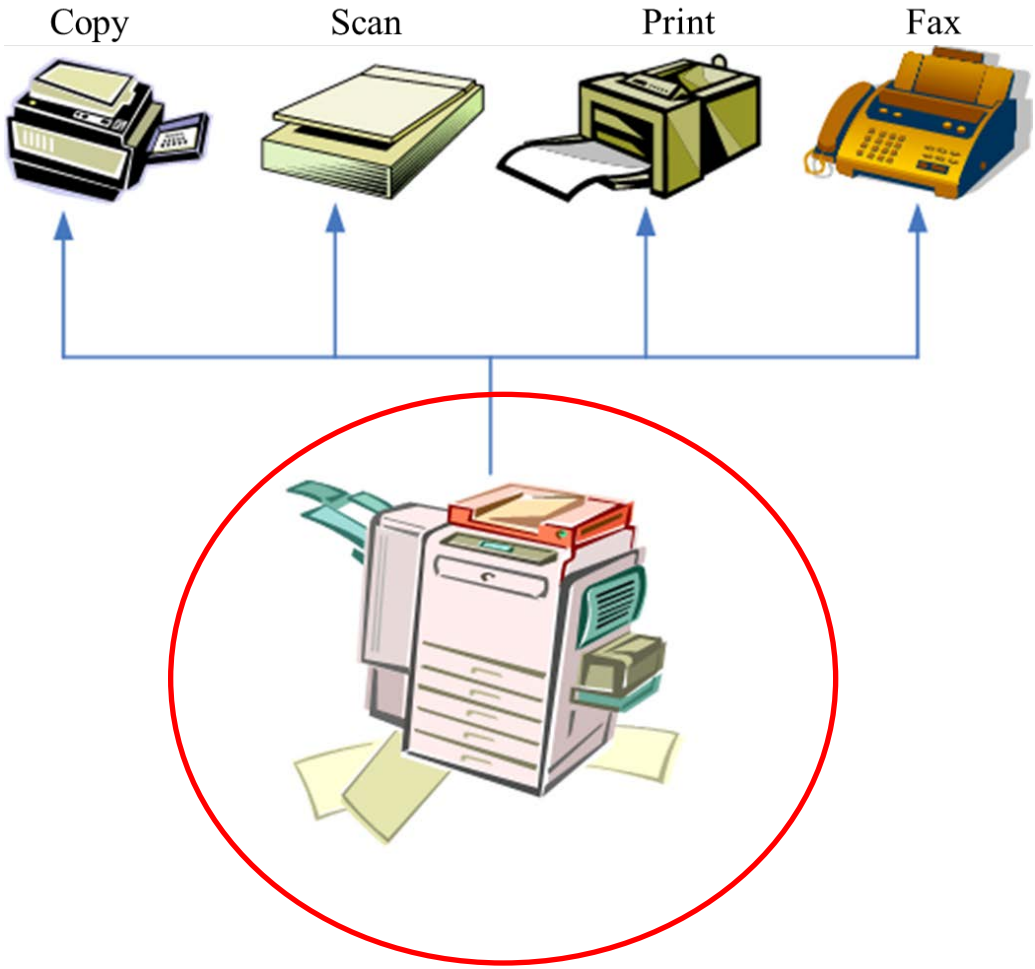
run

void **run**()

オブジェクトが実装するインタフェース Runnable を使ってスレッドを作成し、そのスレッドを開始すると、独立して実行されるスレッド内で、オブジェクトの run メソッドが呼び出されます。

run メソッドの一般的な規約によれば、run メソッドはどのようなアクションを実行してもかまいません。

現実世界に「複合コピー機」という商品が存在する。これは、コピー機としての機能を持ちながら、その他にもスキャナ、ファクシミリ、プリンタなどの機能を持つ機械である。

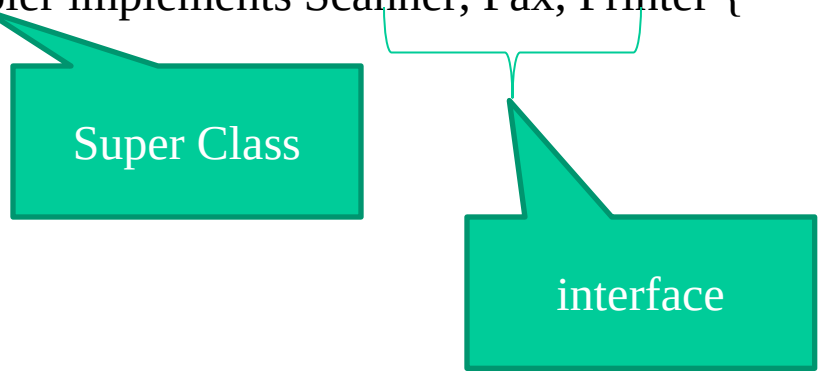


インターフェース宣言

```
public interface Scanner {
    // スキャナの機能
    public abstract Document scan(Paper document);
}
public interface Fax {
    // ファクシミリの機能
    public abstract Paper receive();
    public abstract void send(Paper document, String to);
}
public interface Printer {
    // プリンタの機能
    public abstract Paper print(Document document);
}
```

インターフェース実装

```
public class MultiPurposeCopier extends Copier implements Scanner, Fax, Printer {
    // コピー機の機能
    public Paper copy(Paper origin) {}
    // スキャナの機能
    public Document scan(Paper document) {}
    // ファクシミリの機能
    public Paper receive() ..
    public void send(Paper document, String to) {}
    // プリンタの機能
    public Paper print(Document document) {}
}
```



インターフェースは、上記の複合コピー機ならば「スキャンのボタン」「ファックス送受信のボタン」「プリンタのボタン」というイメージが近い。

参考リンク

http://java2005.cis.k.hosei.ac.jp/materials/lecture19/abstractclass.html

マルチスレッドの同期
Multi-Thread Synchronization

どうしていつかスレッドの同期が必要か？

オブジェクトでは、一度に1個のスレッドしか同期メソッドを実行することができない時。

どうやってマルチスレッド同期することができるか？

同期メソッドとは何か？　＝＞同期メソッドはメソッドをsynchronizedで宣言することです。

モニタオブジェクトとは何か？＝＞同期メソッドを持つオブジェクトはモニタオブジェクトと言う。

モニタオブジェクトに複数の同期メソッドがある場合も、同時には1個の同期メソッドしか実行されません。

同期メソッドを呼び出そうとするほかのスレッドは全て待ったされます。

実行中の同期メソッドが終了するとそのオブジェクトのロックが解除され、

モニタは同時メソッドを呼び出そうと待っているスレッドの中から最も優先度の高いものを実行します。

public final void wait()
public final void notify()
public final void notifyAll()

同期メソッドを実行しているスレッドがそれ以上進めないと判断した時は、自発的にwaitメソッドを呼び出す。

Waitメソッドの呼び出しには、待機状態を終わらせるためのnotifyまたはnotifyAllメソッドの呼び出しが必ず対応していなければいけません。

マルチスレッド化したプログラムを正常に動かすにはスレッドの同期が不可欠ですが、スレッドが待たされるためにプログラムのパフォーマンスが悪くなることがあります。けれども、しょうがないです。

現実世界のケース

https://cis.k.hosei.ac.jp/~rhuang/Miccl/Java3/SushiApplet/index.html

二つの職人スレッド
いくつの客スレッド
➔ 独立並行して動く

例1：マルチスレッドの例

An example of multithreading: **Producer/Consumer**

Without thread synchronization

IntegerStoreの最大値は一つ整数です

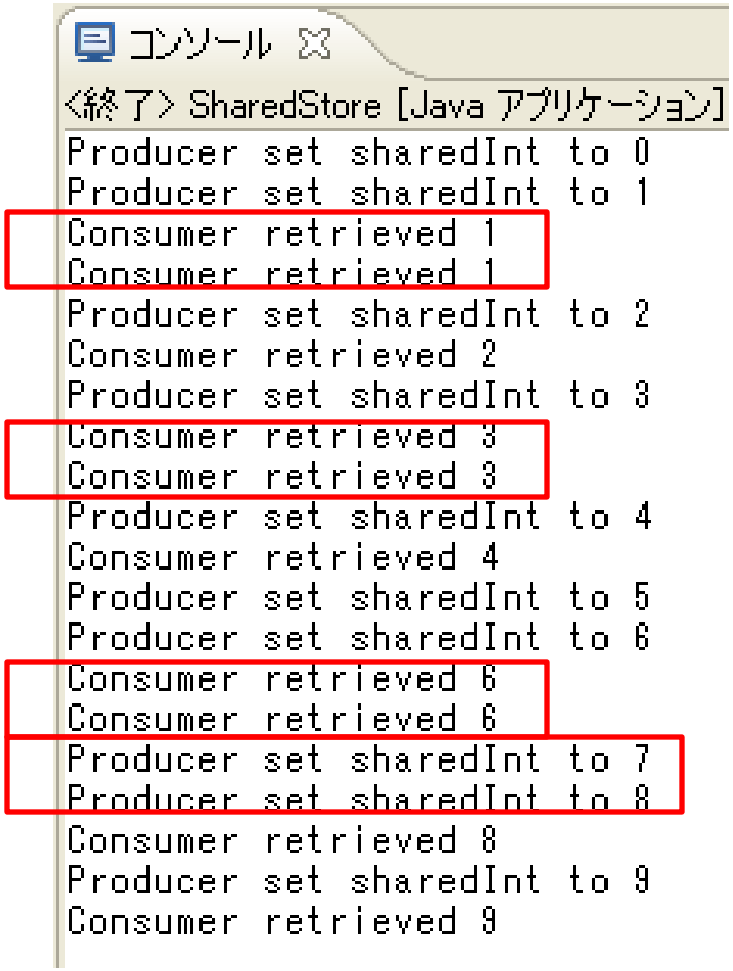
```
class Producer  extends Thread {
  private IntegerStore  pStore;              // 変数
  public Producer( IntegerStore iS )     //  コンストラクタメソッド
  {
    pStore = iS;
  }
  public void run()                          // run() メソッド
  {
    for ( int count = 0; count < 10; count++ ) {
      try {
        Thread.sleep( (int) ( Math.random() * 3000 ) );
      }
      catch( InterruptedException e ) {
        System.err.println( e.toString() );
      }
      pStore.setSharedInt( count );
      System.out.println( "Producer set sharedInt to "
                          + count );
    }
    pStore.setMoreData( false );
  }
}
```

```
class Consumer  extends Thread {
  private IntegerStore cStore;
  public Consumer( IntegerStore iS)
  {
    cStore = iS;
  }
  public void run()
  {
    int val;
    while ( cStore.hasMoreData() ) {
      try {
        Thread.sleep( (int) ( Math.random() * 3000 ) );
      }
      catch( InterruptedException e ) {
        System.err.println( e.toString() );
      }
      val = cStore.getSharedInt();
      System.out.println( "Consumer retrieved " + val );
    }
  }
}
```

```
public class SharedStore {
  public static void main( String args[] )
  {     IntegerStore is = new IntegerStore();
        Producer p = new Producer(is );
        Consumer c = new Consumer( is );
        p.start();      c.start();   }}
```

```
class IntegerStore {
  private int sharedInt = -1;   private boolean moreData = true;
  public void setSharedInt( int val ) { sharedInt = val; }
  public int getSharedInt() { return sharedInt; }
  public void setMoreData( boolean b ) { moreData = b; }
  public boolean hasMoreData() { return moreData; }
}
```

例1の実行結果



どの問題？

例2：マルチスレッドの同期の例

An example of multithreading: **Producer/Consumer**

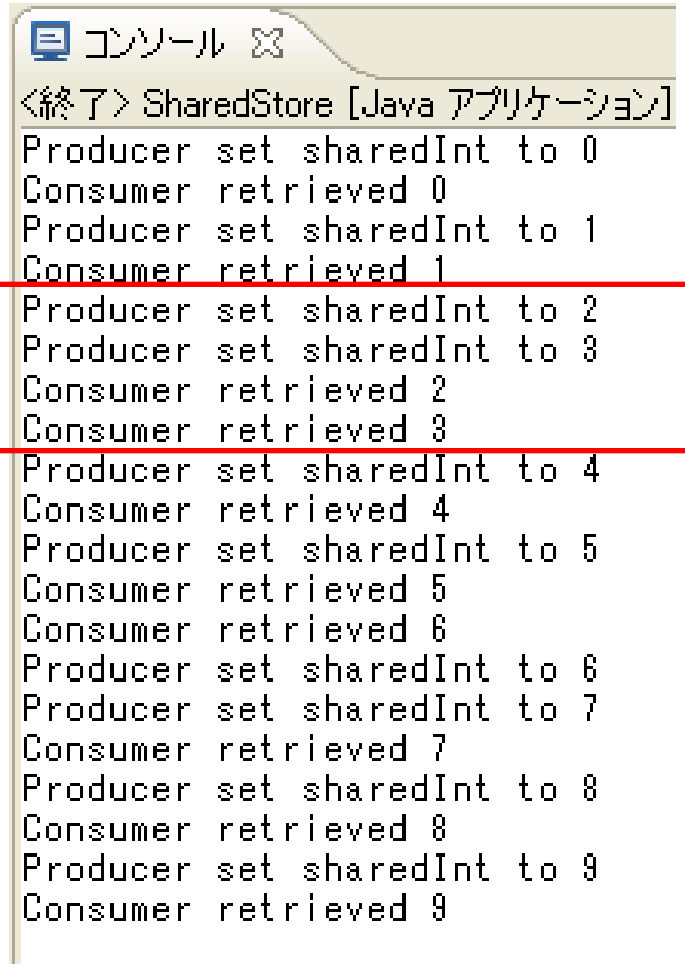
With thread synchronization

IntegerStoreの最大値は一つ整数です

```
class IntegerStore {
   private int sharedInt = -1;
   private boolean moreData = true;
   public void setSharedInt( int val ) { sharedInt = val; }
   public int getSharedInt() { return sharedInt; }
   public void setMoreData( boolean b ) { moreData = b; }
   public boolean  hasMoreData() { return moreData; }
}
```

```
class IntegerStore {
  private int sharedInt = -1;
  private boolean moreData = true;
  private boolean writeable = true;
  public synchronized void setSharedInt( int val ) {
    while(!writeable) {
      try {
        wait();
      }
      catch (InterruptedException e){
        System.err.println("Exception: " + e.toString());
      }
    }
    sharedInt = val;
    writeable = false;
    notify();
  }
  public synchronized int getSharedInt() {
    while (writeable){
      try{
        wait();
      }
      catch(InterruptedException e){
        System.err.println("Exception: " + e.toString());
      }
    }
    writeable = true;
    notify();
    return sharedInt;
  }
  public void setMoreData( boolean b ) { moreData = b; }
  public boolean hasMoreData() { return moreData; }
}
```

例2の実行結果



どの問題？

例1：マルチスレッドの例

An example of multithreading: **Producer/Consumer**

Without thread synchronization

IntegerStoreの最大値は一つ整数です

```
class Producer  extends Thread {
  private IntegerStore  pStore;              // 変数
  public Producer( IntegerStore iS )     //  コンストラクタメソッド
  {
    pStore = iS;
  }
  public void run()                          // run() メソッド
  {
    for ( int count = 0; count < 10; count++ ) {
      try {
        Thread.sleep( (int) ( Math.random() * 3000 ) );
      }
      catch( InterruptedException e ) {
        System.err.println( e.toString() );
      }
      pStore.setSharedInt( count );
      System.out.println( "Producer set sharedInt to "
                          + count );
    }
    pStore.setMoreData( false );
  }
}
```

```
class Consumer  extends Thread {
  private IntegerStore cStore;
  public Consumer( IntegerStore iS)
  {
    cStore = iS;
  }
  public void run()
  {
    int val;
    while ( cStore.hasMoreData() ) {
      try {
        Thread.sleep( (int) ( Math.random() * 3000 ) );
      }
      catch( InterruptedException e ) {
        System.err.println( e.toString() );
      }
      val = cStore.getSharedInt();
      System.out.println( "Consumer retrieved " + val );
    }
  }
}
```

```
public class SharedStore {
  public static void main( String args[] )
  {     IntegerStore is = new IntegerStore();
        Producer p = new Producer(is );
        Consumer c = new Consumer( is );
        p.start();      c.start();   }}
```

```
class IntegerStore {
  private int sharedInt = -1;   private boolean moreData = true;
  public void setSharedInt( int val ) { sharedInt = val; }
  public int getSharedInt() { return sharedInt; }
  public void setMoreData( boolean b ) { moreData = b; }
  public boolean hasMoreData() { return moreData; }
}
```

例2：マルチスレッドの同期の例

An example of multithreading: **Producer/Consumer**
With thread synchronization

```
class IntegerStore {
   private int sharedInt = -1;
   private boolean moreData = true;
   public void setSharedInt( int val ) { sharedInt = val; }
   public int getSharedInt() { return sharedInt; }
   public void setMoreData( boolean b ) { moreData = b; }
   public boolean  hasMoreData() { return moreData; }
}
```

```
class IntegerStore {
  private int sharedInt = -1;
  private boolean moreData = true;
  private boolean writeable = true;
  public synchronized void setSharedInt( int val ) {
    while(!writeable) {
      try {
        wait();
      }
      catch (InterruptedException e){
        System.err.println("Exception: " + e.toString());
      }
    }
    sharedInt = val;
    System.out.println( "Producer set sharedInt to "
                                + val );
    writeable = false;
    notify();
  }
  public synchronized int getSharedInt() {
    while (writeable){
      try{
        wait();
      }
      catch(InterruptedException e){
        System.err.println("Exception: " + e.toString());
      }
    }
    System.out.println( "Consumer retrieved " + sharedInt );
    writeable = true;
    notify();
    return sharedInt;
  }
  public void setMoreData( boolean b ) { moreData = b; }
  public boolean hasMoreData() { return moreData; }
}
```

Work in Class

Run all examples （two examples）
in this lecture notes

演習２

Week1のEx1.（課題１のEx１）には、(in the file, Week1_Ex1)

以下のような、つぎの2つスレッドラスをそれぞれ、方法1で、方法２で）定義せよ。

SquareThread class: （方法１で）

課題１のEx１と同じように

SquareThreadスレッドを生成してスタートさせる。

CubeThread class: （方法２で）

Runnableインタフェースを実装して、

CubeThread クラスにRunnableオブジェクトを生成して、

CubeThreadオブジェクトを生成してスタートさせる。

Javaアプリケーションプログラムを記述せよ。Wee

演習２の出力例

```
Square Thread is started!
Cube Thread is started!
Square Thread    i=1    i*i = 1
Square Thread    i=2    i*i = 4
Square Thread    i=3    i*i = 9
Cube Thread      i=1    i*i*i = 1
Square Thread    i=4    i*i = 16
Cube Thread      i=2    i*i*i = 8
Square Thread    i=5    i*i = 25
Square Thread    i=6    i*i = 36
Cube Thread      i=3    i*i*i = 27
Square Thread    i=7    i*i = 49
Square Thread    i=8    i*i = 64
Cube Thread      i=4    i*i*i = 64
Cube Thread      i=5    i*i*i = 125
Cube Thread      i=6    i*i*i = 216
Square Thread    i=9    i*i = 81
Square Thread is done!
Cube Thread      i=7    i*i*i = 343
Cube Thread      i=8    i*i*i = 512
Cube Thread      i=9    i*i*i = 729
Cube Thread is done!
```

先週の課題1（参考用）

課題1

Ex1. 以下のような、つぎの2つのスレッドクラスを定義せよ。

SquareThread class:

スレッドの名前をセットするコンストラクタ

for i=1～9,

iの値とスレッドの名前、i*iの値を0～3000ミリ秒のスリープタイムで出力する。

スレッドの名前と文字列"is done!"を出力する。

CubeThread class:

スレッドの名前をセットするコンストラクタ

for i=1～9

iの値とスレッドの名前、i*i*iの値を0～3000ミリ秒のスリープタイムで出力する。

スレッドの名前と文字列“is done!”を出力する。

SquareThreadスレッドとCubeThreadスレッドを生成してスタートさせるJavaアプリケーションプログラムを記述せよ。

(ヒント: Count10ThreadUp　Count10ThreadDownを参照)

Ex1の出力例

```
Square Thread is started!
Cube Thread is started!
Square Thread    i=1    i*i = 1
Square Thread    i=2    i*i = 4
Square Thread    i=3    i*i = 9
Cube Thread      i=1    i*i*i = 1
Square Thread    i=4    i*i = 16
Cube Thread      i=2    i*i*i = 8
Square Thread    i=5    i*i = 25
Square Thread    i=6    i*i = 36
Cube Thread      i=3    i*i*i = 27
Square Thread    i=7    i*i = 49
Square Thread    i=8    i*i = 64
Cube Thread      i=4    i*i*i = 64
Cube Thread      i=5    i*i*i = 125
Cube Thread      i=6    i*i*i = 216
Square Thread    i=9    i*i = 81
Square Thread is done!
Cube Thread      i=7    i*i*i = 343
Cube Thread      i=8    i*i*i = 512
Cube Thread      i=9    i*i*i = 729
Cube Thread is done!
```

課題2

How to change it to 回転すし
（1）一人職人
（2）一人客
（3）テーブルsize = 1

Ex1. Motherがa sequence of foodsを与えると仮定せよ。i=0～9で、Babyは以下に従う。

Mother takes 1000~3000ms to feed

Baby takes 1000~4000 to eat

mouthの最大値は一つfeed

1.1 Mother feedingとBaby eatingに同期するsynchronized feed(int value)メソッドとsynchronized eat(int value)メソッドを持つ**Mouth**ラスを定義せよ。*(Refer **IntegerStore** class)*

1.2 以下のようなMotherThread classとBabyThread classを定義せよ。

MotherThread class: *(Refer **Producer** class)*

スレッドの名前をセットするコンストラクタ

for i=0～9,

スレッドの名前、iのfeeding sequenceをプリントする。

BabyThread class: *(Refer **Consumer** class)*

スレッドの名前をセットするコンストラクタ

for i=0～9,

スレッドの名前、iのeating sequenceをプリントする。

1.3 MotherThreadスレッドとBabyThreadスレッドを生成してスタートさせるJavaアプリケーションプログラム(**Ex3**)を記述せよ。*(Refer **ShareStore**)*

```
class Mouth{
    food = -1;
    boolean feedMore=false, feed=true, eat=false;
    public synchronized void feed( int val ) { ; }
    public synchronized int eat() { return -2; }
    public void feedMore( boolean b ) { ; }
    public boolean hasFood() { return false; }
}
class MotherThread extends Thread {} //constructor, run()

class BabyThread extends Thread { } //constructor, run()

public class Ex3 { } //main method, application program
```