จาวา : Threads

สมชาย ประสิทธิ์จูตระกูล

Threads

- A thread == A virtual CPU
- Threads == Virtual CPUs
  - sharing the same memory space
  - running “virtually” at the same time
- Java supports multithreading
- Typically used for
  - increasing UI response
  - animation
  - networking

```
public class Ball {
  int x, y, r, dx = rand(-5, 5), dy = rand(-5, 5);
  BallWorld world;
  Color color = ColorUtil.random();
  public Ball(BallWorld ballWorld) {
    x = rand(0, ballWorld.getWidth());
    y = rand(0, ballWorld.getHeight());
    r = rand(2, 10);
    world = ballWorld;
  }
  public void move() {
    int ww = world.getWidth(), wh = world.getHeight();
    x = x + dx,  y = y + dy;
    int d = 2 * r;
    if (x + d >= ww) { x = ww - d; dx = -dx; }
    if (x <= 0) { x = 0; dx = -dx; }
    if (y + d >= wh) { y = wh - d; dy = -dy; }
    if (y <= 0) { y = 0; dy = -dy; }
  }
  public void draw(Graphics g) {
    g.setColor(color);
    g.fillOval(x, y, 2 * r, 2 * r);
  }
}
```

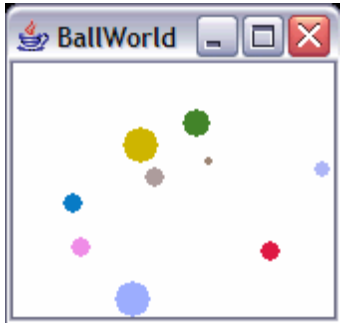
สุ่มตำแหน่งและขนาด

เลื่อนลูกบอลกลับทิศถ้าชนขอบ

วาดลูกบอล

```
public class BallWorld extends Canvas {
  List balls = new ArrayList();

  public void init() {
    for (int i = 0; i < 10; i++) {
      balls.add(new Ball(this));
    }
    while (true) {
      for (int i = 0; i < balls.size(); i++) {
        ((Ball) balls.get(i)).move();
      }
      repaint();
      try { Thread.sleep(67); }
      catch (InterruptedException ignored) {}
    }
  }
  public void paint(Graphics g) {
    super.paint(g);
    for (int i = 0; i < balls.size(); i++) {
      ((Ball) balls.get(i)).draw(g);
    }
  }
```



ควบคุมการเคลื่อนไหวของบอลทุกลูก

ควบคุมการแสดงออกจอ (15 ภาพ / s)

```
public class BallWorld extends Canvas {
  List balls = new ArrayList();

  public void init() {
    for (int i = 0; i < 5; i++) {
      balls.add(new Ball(this));
      balls.add(new SBall(this));
    }
    while (true) {
      for (int i = 0; i < balls.size(); i++) {
        Ball b = (Ball) balls.get(i);
        b.move();

        if (b instanceof SBall) b.split();

      }
      repaint();
      try { Thread.sleep(67); }
      catch (InterruptedException ignored) {}
    }
  }
  ...
```

เพิ่มลูกบอลที่แตกตัวได้

ควบคุมการเคลื่อนไหว ของบอลทุกลูก

ถ้าเป็น SBall ให้แตกตัว

ควบคุมการ แสดงออกจอ (15 ภาพ / s)

```
public class BallWorld extends Canvas {
  List balls = new ArrayList();
  public void init() {
    for (int i = 0; i < 5; i++) {
      balls.add(new Ball(this));
      balls.add(new SBall(this));
    }
    while (true) {
      repaint();
      try { Thread.sleep(67); }
      catch (InterruptedException ignored) {}
    }
  }
  ...
```

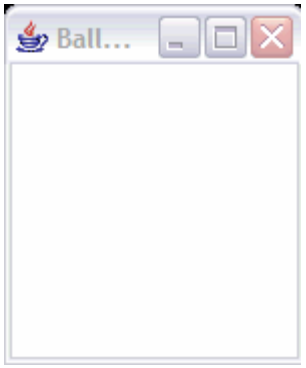
ถ้าบอลแต่ละลูก ทำงานเองได้ ก็ให้เขาเลื่อนและเปลี่ยนแปลงตามพฤติกรรมที่ตัวเองอยากเป็น

หนึ่งลูก หนึ่ง thread ทำงานกันพร้อมๆ กันไป

```
public class Ball implements Runnable {
  public Ball(BallWorld ballWorld) {
    ...   new Thread(this).start();
  }
  public void run() {
    while (true) {
      move();
      try { Thread.sleep(100); }
      catch (InterruptedException ignored) {}
    }
  }
  ...
```

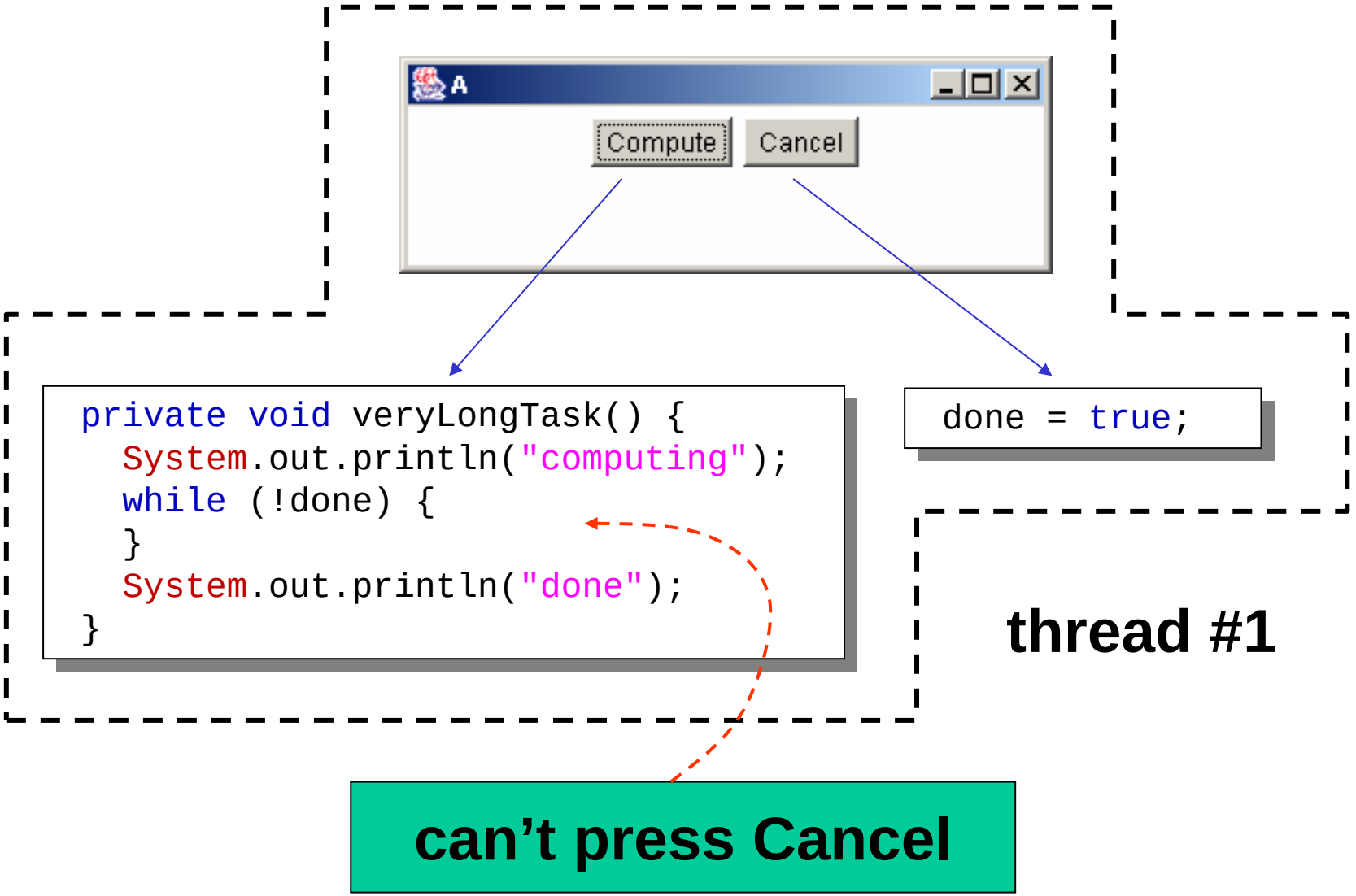
```
public class SBall extends Ball {
  public SBall(BallWorld ballWorld) {
    super(ballWorld);
  }
  public SBall(SBall b) {
    super(b);
  }

  public void run() {
    int longevity = 10 + (int) (10 * Math.random());
    for (int i = 0; i < longevity; i++) {
      move();
      try { Thread.sleep(100); }
      catch (InterruptedException ignored) {}
    }
    world.balls.remove(this);
    int n = 2 + (int) (2 * Math.random());
    for (int i = 0; i < n; i++) {
      world.balls.add(new SBall(this));
    }
  }
}
```

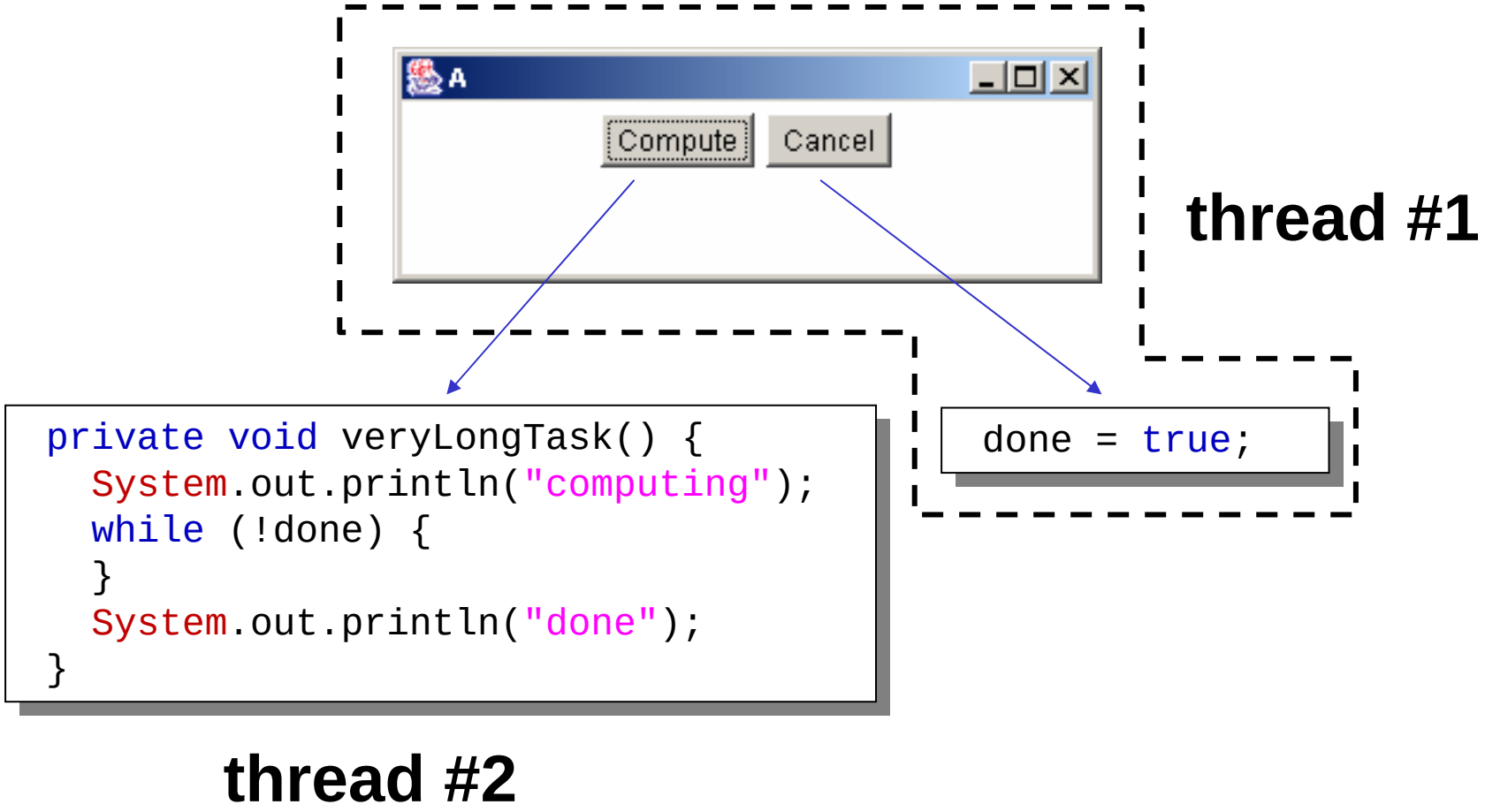


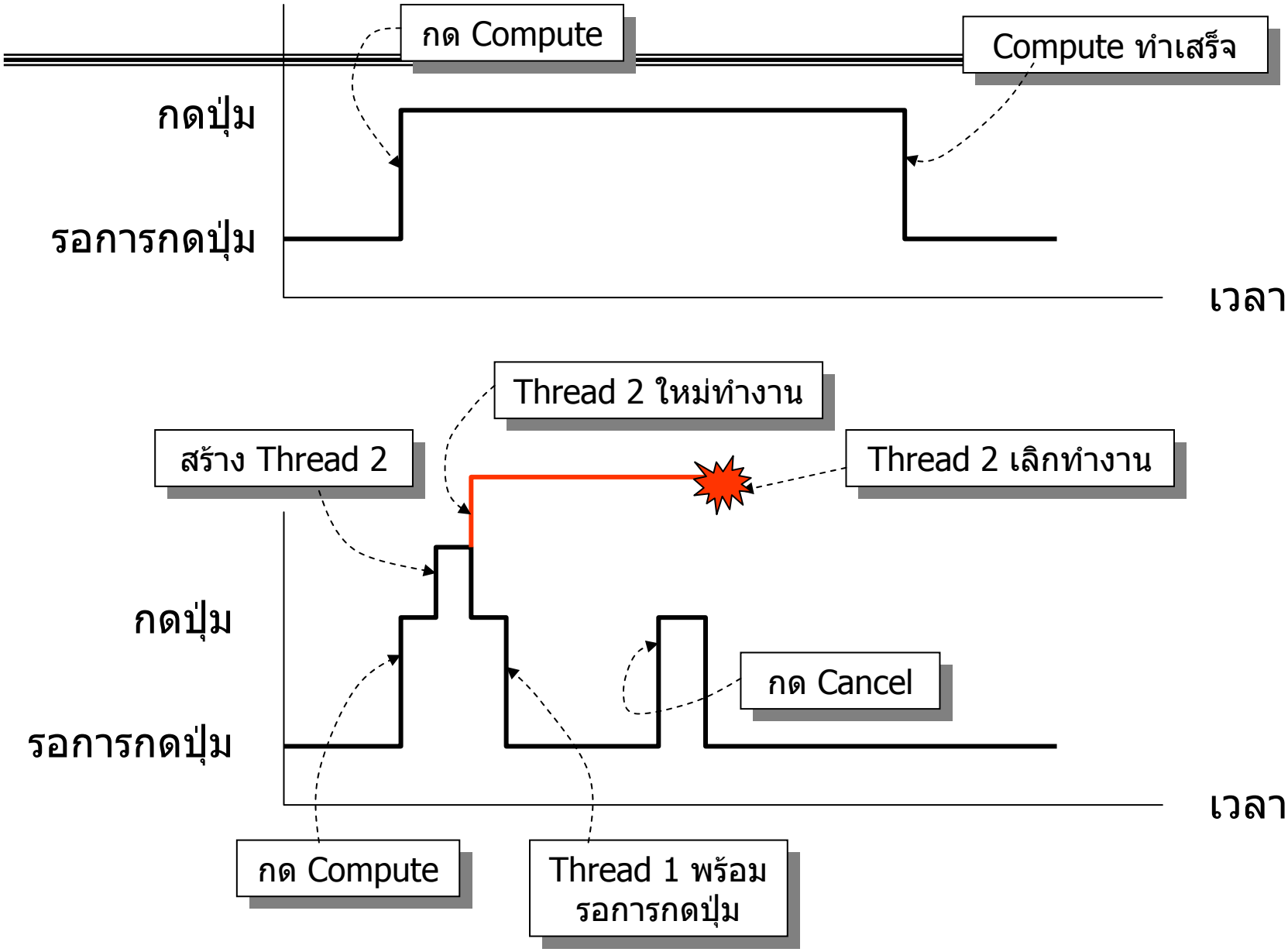
ยังมีปัญหาเรื่อง sync.

An Example


A
Compute
Cancel
private void veryLongTask() {
System.out.println("computing");
while (!done) {
}
System.out.println("done");
}
done = true;
thread #1
can’t press Cancel

An Example


A
Compute
Cancel
thread #1
private void veryLongTask() {
System.out.println("computing");
while (!done) {
}
System.out.println("done");
}
done = true;
thread #2


กด Compute
Compute ทำเสร็จ
กดปุ่ม
รอการกดปุ่ม
เวลา
Thread 2 ใหม่ทำงาน
สร้าง Thread 2
Thread 2 เลิกทำงาน
กดปุ่ม
กด Cancel
รอการกดปุ่ม
เวลา
กด Compute
Thread 1 พร้อม
รอการกดปุ่ม

การสร้าง thread : แบบที่ 1

```
class MyThread extends Thread {

   public void run() {

      doSomething();
   }

   ...
}
```

this run() overrides Thread's run()

put your execution code here

```
...
Thread th = new MyThread();
th.start();
...
```

create a new object

invoke start()
- creates new thread,
- makes it runnable,
- and then returns immediately.

The new thread
- starts execution at its run() method
- terminates when run() exits.

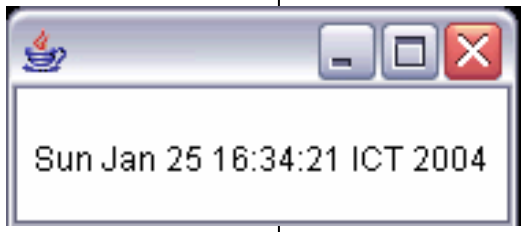
ตัวอย่างการสร้าง thread

```
public class A {
  public static void main(String[] args) {
    T1[] threads = new T1[4];
    for (int i = 0; i < threads.length; i++) {
      threads[i] = new T1();
      threads[i].start();
    }
  }
}
```

```
class T1 extends Thread {
  public void run() {
    for (int i = 0; i < 100; i++) {
      System.out.println(getName() + ":" + i);
      for (int j=0; j<1000000; j++) j = j;
    }
  }
}
```

```
Thread-0:0
Thread-0:1
Thread-0:2
Thread-1:0
Thread-1:1
Thread-0:3
Thread-0:4
Thread-1:2
Thread-1:3
Thread-2:0
...
```

```
class TimerThread extends Thread {
  Frame owner;  long duration;
  TimerThread(Frame owner, long duration) {
    this.owner = owner; this.duration = duration;
    start();
  }
  public String getDate() { return new Date().toString(); }
  public void run() {
    while (true) {
      try { Thread.sleep(duration); }
      catch (InterruptedException ignored) {}
      owner.repaint();
    }
  }
}
```



ไม่สวย !

```
class ClockDemo extends Frame {
  TimerThread clock;
  void init() {
    clock = new TimerThread(this, 1000);
    setSize(200, 100);  setVisible(true);
  }
  public void paint(Graphics g) {
    g.drawString(clock.getDate(), 10, 60);
  }
  public static void main(String[] args) {
    new ClockDemo().init();
  }
}
```

```
class TimerThread extends Label {
  long duration;
  TimerThread(long duration) {
    this.duration = duration;

    new Thread( this ).start();

  }
  public void paint(Graphics g) {
    g.clearRect(0, 0, getWidth(), getHeight());
    g.drawString(new Date().toString(), 10, 20);
  }
  public void run() {
    while (true) {
      try { Thread.sleep(duration); }
      catch (InterruptedException ignored) {}
      repaint();
    }
  }
}
```

มั่นใจได้อย่างไรว่ามี run( ) ให้ thread เรียกตอน start

```
public class ClockDemo extends Frame {
  void init() {
    add(new TimerThread(1000));
    setSize(200, 80);
    setVisible(true);
  }
  public static void main(String[] args) {
    new ClockDemo().init();
  }
}
```

การสร้าง Thread : แบบที่ 2

```
class MyClass implements Runnable {

   public void run() {

      doSomething();
   }

   ...
}
```

this run() implements run() of Runnable interface

put your execution code here

```
MyClass x = new MyClass();
...
Thread th = new Thread(x);
th.start();
...
```

create a new object

start()
- creates new thread,
- and then returns immediately.

The new thread
- starts execution at x's run() method
- terminates when x's run() exits.

```
public interface Runnable{
  public void run();
}
```

```
class TimerThread extends Label implements Runnable {
  long duration;
  TimerThread(long duration) {
    this.duration = duration;

    new Thread( this ).start();

  }
  public void paint(Graphics g) {
    g.clearRect(0, 0, getWidth(), getHeight());
    g.drawString(new Date().toString(), 10, 20);
  }
  public void run() {
    while (true) {
      try { Thread.sleep(duration); }
      catch (InterruptedException ignored) {}
      repaint();
    }
  }
}
```

มั่นใจได้แน่อนว่ามี run( ) ให้ thread เรียกตอน start

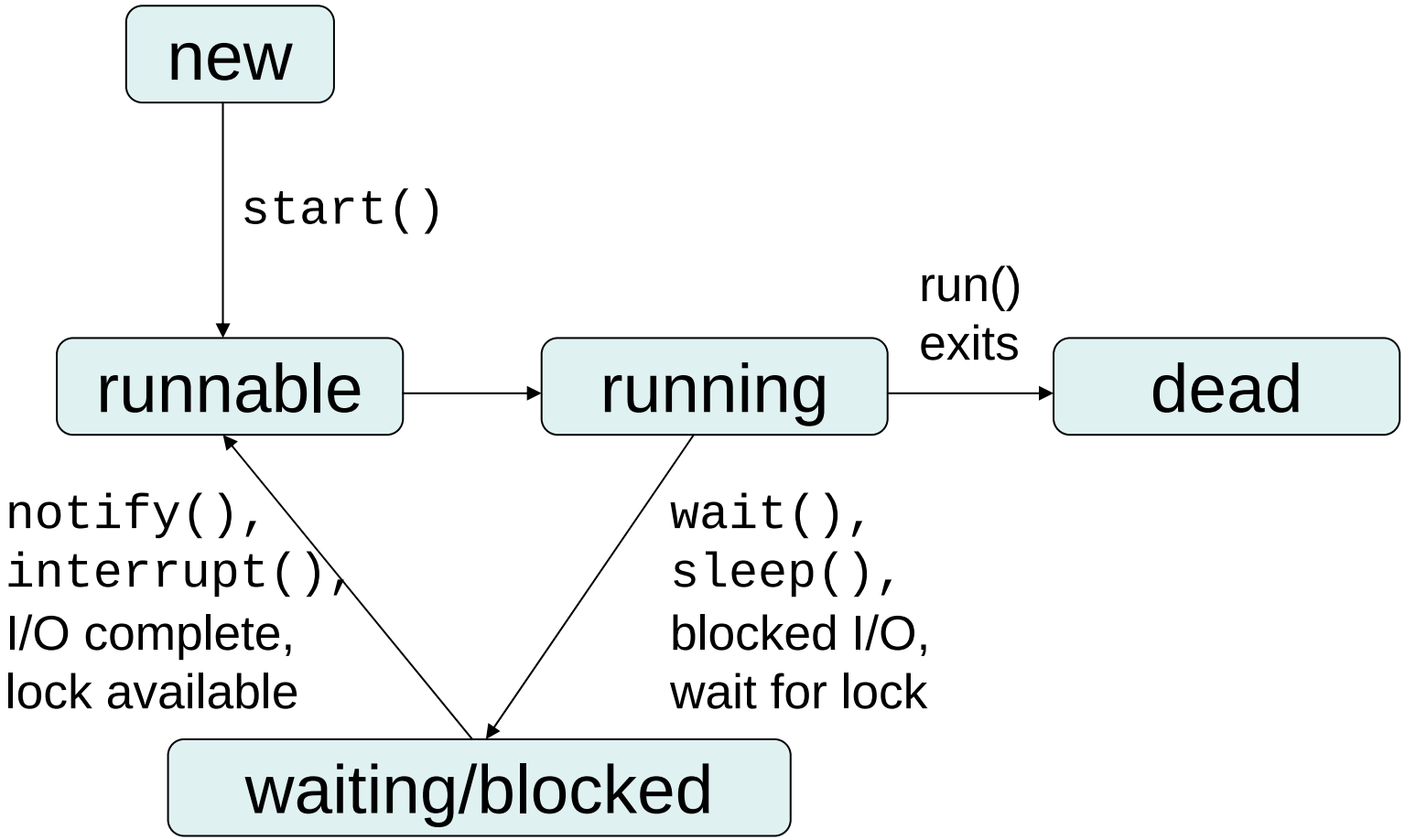
```
public class ClockDemo extends Frame {
  void init() {
    add(new TimerThread(1000));
    setSize(200, 80);
    setVisible(true);
  }
  public static void main(String[] args) {
    new ClockDemo().init();
  }
}
```

A Thread can only be started once

```
public class A {
  public static void main(String[] args) {
    T1 t = new T1();
    t.start();
    t.start(); // t is alive, throw IllegalThreadStateException
    while (t.isAlive()) ;
    t.start(); // return immediately, can't restart dead threads
  }
}
```

```
class T1 extends Thread {
  public void run() {
    for (int i = 0; i < 100; i++) {
      System.out.println(getName() + ":" + i);
      for (int j=0; j<1000000; j++) j = j;
    }
  }
}
```

Thread States



การรอให้ thread ทำงานเสร็จ

```
void initialize() {
  Thread thDB = new Thread(new DBConnection());
  Thread thImage = new Thread(new BannerImage());
  thDB.start();
  thImage.start();
  while (thDB.isAlive());
  while (thImage.isAlive());
}
```

```
void initialize() {
  Thread thDB = new Thread(new DBConnection());
  Thread thImage = new Thread(new BannerImage());
  thDB.start();
  thImage.start();
  try { thDB.join(); thImage.join(); }
  catch( InterruptedException ignored ) { }
}
```

Money Transfer Simulation

```
class Account {
  int amount;
  public Account(int a) {
    amount = a;
  }
  public void deposit(int d) {
    amount += d;
  }
  public void withdraw(int w) {
    amount -= w;
  }
  public int current() {
    return amount;
  }
}
```

```
class Bank {
  Account[] accounts;
  Bank(int n, int amount) {
    accounts = new Account[n];
    for (int i = 0; i < accounts.length; i++)
      accounts[i] = new Account(amount);
  }
  public void transfer(int from, int to, int amount) {
    if (accounts[from].current() >= amount) {
      accounts[from].withdraw(amount);
      accounts[to].deposit(amount);
    }
  }
  public int getTotal() {
    int sum = 0;
    for (int i = 0; i < accounts.length; i++) {
      sum += accounts[i].current();
    }
    return sum;
  }
}
```

```
public class AccountSimulation {
  public static void main(String[] args) {
    final int numAccounts = 4;
    final int numThreads = 10;
    final int initialAmount = 100;
    final Bank myBank = new Bank(numAccounts,
                                 initialAmount);
    Thread[] t = new Thread[numThreads];
    for (int i = 0; i < t.length; i++) {
      t[i] = new Thread() {
        public void run() {
          while (true) {
            int i = (int)(numAccounts * Math.random());
            int j = (int)(numAccounts * Math.random());
            int amount = (int)(50 * Math.random());
            if (i != j) myBank.transfer(i, j, amount);
          }
        }
      };
      t[i].start();
    }
```

Four threads randomly transfer money among 4 accounts

```
Thread sumThread = new Thread() {
  public void run() {
    int sum;
    while (true) {
      sum = myBank.getTotal();
      if (sum != numAccounts * initialAmount) {
        myBank.dumpAccounts();
        System.exit(-1);
      }
    }
  }
};
sumThread.start();
```

Total amount of all accounts must be the same all the time.

Race Condition

```
class Account {
  int amount;
  ...
  public void deposit(int d) {
    amount += d;
  }
}
```

is not atomic

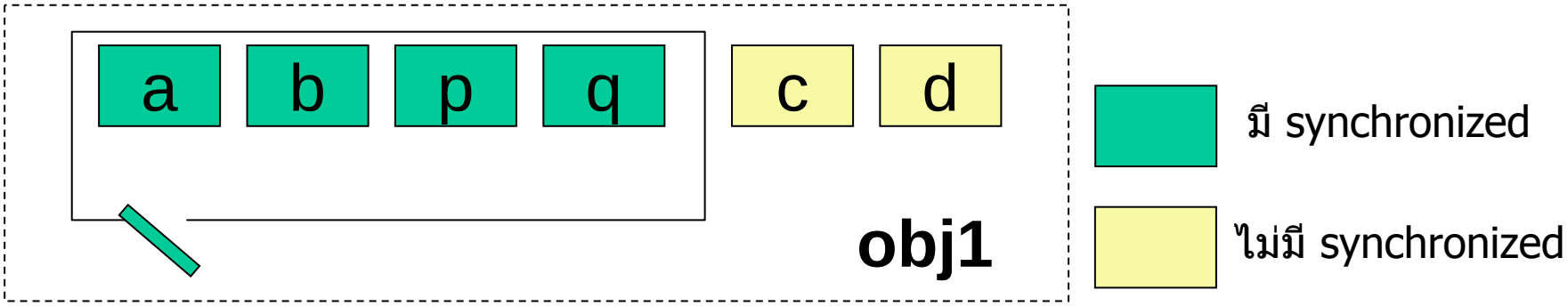
```
0:    aload_0
1:    dup
2:    getfield #3;
5:    iload_1
6:    iadd
7:    putfield #3;
10:   return
```

Thread-1: acc[2].deposit(10);
Thread-2: acc[2].deposit(50);

```
reg1 <- amount
reg1 <- reg1 + d
amount <- reg1
```

Thread-1: reg1 <- amount // 200
Thread-2: reg1 <- amount // 200
Thread-2: reg1 <- reg1 + d // d = 50
Thread-2: amount <- reg1 // 250
Thread-1: reg1 <- reg1 + d // d= 10
Thread-1: amount <- reg1 // 210

Synchronized Object Methods



thread ใดที่ต้องการเรียก obj1. ต้องได้ lock ของ obj1 ก่อน

หนึ่ง object มีเพียงหนึ่ง lock ดังนั้นระบบอนุญาตให้เพียงหนึ่ง thread เท่านั้นที่เรียก obj1. ได้

```
class A {
  synchronized void a() {...}
  synchronized void b() {...}
  void c() {...}
}
```

```
A obj1 = new A();
A obj2 = new A();
```

```
obj1.a();
obj1.c();
obj2.a();
```

```
obj1.a();
obj1.b();
obj2.c();
```

synchronized

```
class A {
  synchronized void a() {...}
  synchronized void b() {...}
  void c() {...}
}
```

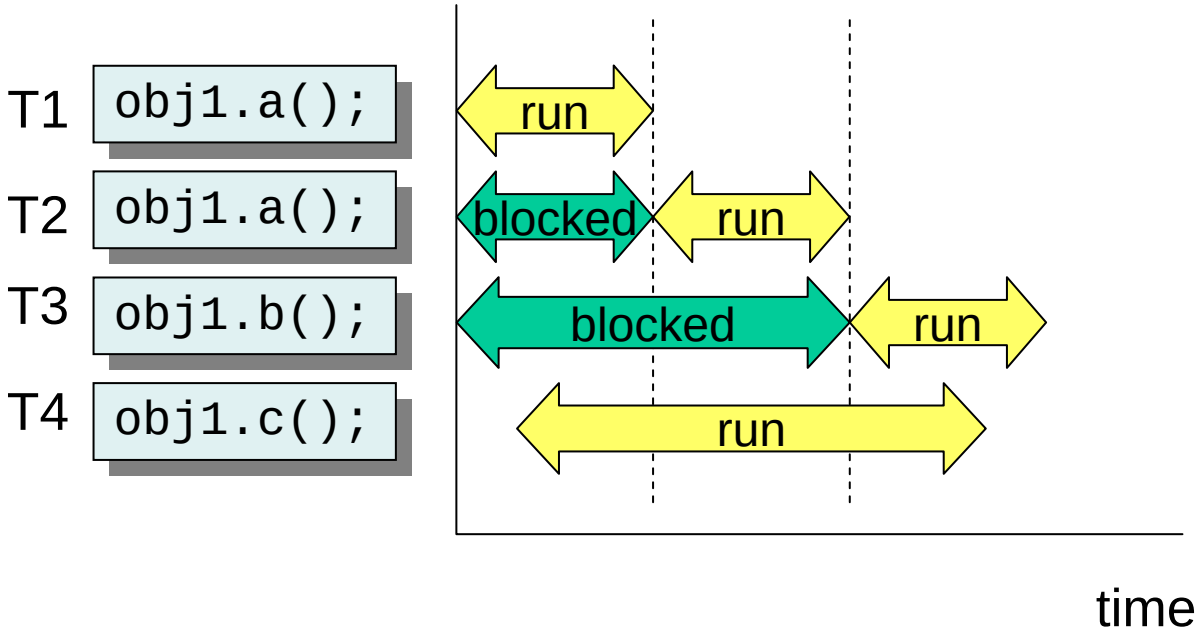
```
A obj1 = new A();
A obj2 = new A();
```

T1	`obj1.a();`	get obj1's lock, can exclusively execute obj1.a()
T2	`obj2.a();`	get obj2's lock, can exclusively execute obj2.a()
T3	`obj1.a();`	blocked, wait for obj1's lock
T4	`obj2.b();`	blocked, wait for obj2's lock
T5	`obj1.c();`	no need to get any lock, go ahead for obj1.c()
T6	`obj1.c();`	no need to get any lock, go ahead for obj1.c()

synchronized

```
class A {
  synchronized void a() {...}
  synchronized void b() {...}
  void c() {...}
}
```

```
A obj1 = new A();
```



synchronized

```
class Bank {
  Account[] accounts;
  ...
  public synchronized void transfer(int from, int to,
                                    int amount) {
    if (accounts[from].current() >= amount) {
      accounts[from].withdraw(amount);
      accounts[to].deposit(amount);
    }
  }
  public synchronized int getTotal() {
    int sum = 0;
    for (int i = 0; i < accounts.length; i++) {
      sum += accounts[i].current();
    }
    return sum;
  }
}
```

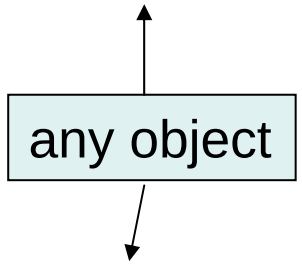
ก่อนเข้าต้องได้ lock ของ bank ที่เรียก

sync. ทำให้ธนาคารนี้มีช่องฝาก-ถอนช่องเดียว !!

synchronized block

```
class A {
  synchronized void a() {
    ...
  }
  void d() {
    synchronized (this) {
      ...
    }
  }
  void e() {
    ...
    synchronized (this) {
      ...
    }
    ...
  }
}
```

```
class A {
  synchronized void a() {
    ...
  }
  void d() {
    synchronized (obj9) {
      ...
    }
  }
  void e() {
    ...
    synchronized (ok) {
      ...
    }
    ...
  }
}
```



synchronized block

```
class Bank {
  Account[] accounts;
  public synchronized void transfer(int from, int to,
                                    int amount) {
    if (accounts[from].current() >= amount) {
      accounts[from].withdraw(amount);
      accounts[to].deposit(amount);
    }
```

lock the entire bank

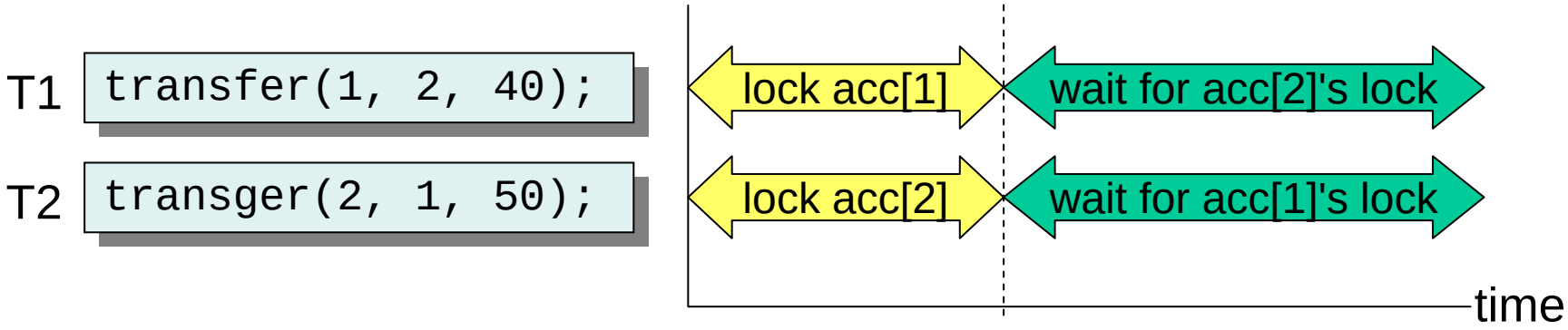
```
class Bank {
  Account[] accounts;
  public void transfer(int from, int to, int amount) {
    synchronized (accounts[from]) {
      synchronized (accounts[to]) {
        if (accounts[from].current() >= amount) {
          accounts[from].withdraw(amount);
          accounts[to].deposit(amount);
        }
      }
    }
  }
```

lock only 2 accounts

more efficient & scalable

Deadlock

```
class Bank {
  Account[] accounts;
  public void transfer(int from, int to, int amount) {
    synchronized (accounts[from]) {
      synchronized (accounts[to]) {
        if (accounts[from].current() >= amount) {
          accounts[from].withdraw(amount);
          accounts[to].deposit(amount);
        }
      }
    }
  }
```



Deadlock

```
class Bank {
  Account[] accounts;
  public void transfer(int from, int to, int amount) {
    int i = (from < to ? from : to);
    int j = (from < to ? to : from);
    synchronized (accounts[i]) {
      synchronized (accounts[j]) {
        if (accounts[from].current() >= amount) {
          accounts[from].withdraw(amount);
          accounts[to].deposit(amount);
        }
      }
    }
  }
```

ordering the locking sequence
(always lock account with smaller ID first)

There is no known techniques to detect or prevent deadlock.

Notes

- any variable assignment is atomic except for `long` and `double`
- synchronization isn't free
- no promises about fairness
- use immutable objects
- use higher level concurrency abstraction
  - JSR-166 (concurrency utilities) included in Java 1.5