**JAVA\_MULTITHREAD**

1. **Mục đích của Multithread.**

* Tăng hiệu suất của CPU.
* Xử lí những tác vụ khác trong khi chờ một tác vụ nào đó nghỉ.
* Tăng tốc độ ứng dụng.

1. **Sử dụng Multithread trong java.**

* Để sử dụng Multithread trong java chúng ta có thể sử dụng 2 cách sau đây.
* Cách 1: Extends class Thread.
  + Ưu điểm: trở thành lớp con của Thread và sử dụng các hàm của nó.
  + Nhược điểm: Không thể extend thêm một lớp nào khác vì cơ chế đa kế thừa trong java không tồn tại.
* Cách 2: Implement class Runable.
  + Ưu điểm: có thể sử dụng implement nhiều Interface khác nhau và Extends một lớp khác.
  + Nhược diểm: phải implement lại các phương thức.
* **VD:** viết chương trình thể hiện luồng bằng cách tạo ra hai luồng độc lập Th1 và Th2 (Th1 và Th2 có tác dụng in ra các số từ 1 đến 99). Khởi động chương trình và xem kết quả và đưa ra nhận xét.
  + **Sử dụng cách 1:**

|  |
| --- |
| class Runner extends Thread  {      public void run() // chú ý phương thức run() này chúng ta đang Override lại nó trong lớp Thread chứ không phải là một phương thức riêng của Runner.      {          for (int i = 0; i < 100; i++)          {              System.out.println(i);          }      }  }  class test\_Thread  {      public static void main(String[] args)      {          Thread th1 = new Thread(new Runner()); // tạo đối tượng của lớp Thread thể hiện sang đối tượng Runner.          Thread th2 = new Thread(new Runner());          th1.start(); // động tác này là khởi động luồng, khi khởi động luồng thì nó sẽ tìm hàm run() trong luồng và chạy.          th2.start();      }  }  **Như chúng ta thấy thì kết quả của nó không như chúng ta mong muốn các luồng chạy xen kẽ và độc lập với nhau.** |

* + **Sử dụng cách 2:**

|  |
| --- |
| class Runner implements Runnable  {      public void run() // chú ý phương thức run() này chúng ta đang implements cho phương thức này trong lớp Runnable chứ không phải là một phương thức riêng của Runner.      {          for (int i = 0; i < 100; i++)          {              System.out.println(i);          }      }  }  class test\_Thread  {      public static void main(String[] args)      {          Thread th1 = new Thread(new Runner()); // tạo đối tượng của lớp Thread thể hiện sang đối tượng Runner.          Thread th2 = new Thread(new Runner());          th1.start(); // động tác này là khởi động luồng, khi khởi động luồng thì nó sẽ tìm hàm run() trong luồng và chạy.          th2.start();      }  } |

* **Bài tập:**

Tạo một chương trình giữa người sản xuất và người mua. Người sản xuất sẽ cứ sản xuất các mặt hàng và cho vào trong kho. Người mua sẽ cứ tiếp tục mua gói hàng đầu tiên ở bên trong kho của người sản xuất và mỗi khi mua thì mặt hàng đó sẽ mất khỏi kho chương trình dừng lại nếu như người dùng yêu cầu mua hàng nhưng lại không có hàng.

* Lớp Product.

|  |
| --- |
| // Lớp này chỉ là một cấu trúc dữ liệu.  public class Product  {      private int Number;      private boolean check;      public Product(int x)      {          this.Number = x;          this.check = true;      }      public void mark\_bought\_product()      {          this.check = false;      }      public boolean check\_bough\_product()      {          return this.check;      }  } |

* Lớp Producer.

|  |
| --- |
| import java.util.ArrayList;  import java.util.Random;  public class Producer implements Runnable  {      private ArrayList<Product> Kho;      private Product obj;      private int i = 0;      private int Number\_Of\_Bought\_Product;      public int Ramdom\_Product()      {          Random rd = new Random();          return rd.nextInt(this.Kho.size());      }      public Producer(Producer obj1)      {          this.Kho = obj1.Kho;          this.Number\_Of\_Bought\_Product = 0;      }      public Producer()      {          this.Kho = new ArrayList<Product>();          this.Number\_Of\_Bought\_Product = 0;      }      public boolean check\_run\_out\_of\_Product()      {          if (this.Number\_Of\_Bought\_Product > this.Kho.size())          {              return false;          }          return true;      }      public void produce(int x)      {          System.out.println("Kho da san xuat " + x);          this.obj = new Product(x);          this.Kho.add(obj);      }      public void remove\_Product(int x)      {          this.obj = new Product(x);          obj.mark\_bought\_product();          this.Kho.set(x, obj);          this.Number\_Of\_Bought\_Product++;      }      public void initialize()      {          produce(i);          i++;      }      public void run()      {          while (true)          {              if (Customer.Flag == 1)              {                  System.out.println("Khach yeu cau nhung het hang");                  return;              }              produce(i);              i++;              try {                  Thread.sleep(500);              } catch (InterruptedException e) {                  // TODO Auto-generated catch block                  System.out.println("Loi");                  System.out.println(e);              }          }      }      public boolean check\_bought\_Product(int Index)      {          return this.Kho.get(Index).check\_bough\_product();      }  } |

* Lớp customer

|  |
| --- |
| import java.util.Random;  public class Customer implements Runnable  {      Producer obj;      public static int Flag;      public Customer(Producer obj)      {          this.obj = obj;          this.Flag = 0;      }      public void buy()      {          int x = this.obj.Ramdom\_Product();          if (this.obj.check\_bought\_Product(x) == true)          {              System.out.println("da mua san pham " + x);              this.obj.remove\_Product(x);          }          else          {              System.out.println("San pham " + x + " khong con");          }      }      public void run()      {          while (true)          {              if (this.obj.check\_run\_out\_of\_Product() == false)              {                  System.out.println("Kho het hang");                  this.Flag = 1;                  return;              }              buy();              try {                  Thread.sleep(1000);              } catch (InterruptedException e) {                  // TODO Auto-generated catch block                  System.out.println("Loi");                  e.printStackTrace();              }          }      }  } |

* Lớp Main

|  |
| --- |
| import java.util.ArrayList;  import java.util.Random;  class Main  {      public static void main(String[] args)      {          Producer obj1 = new Producer();          obj1.initialize();          Customer cus = new Customer(obj1);          Thread th1 = new Thread(obj1);          Thread th2 = new Thread(cus);            th1.start();          th2.start();      }  } |

**Gom nhóm các đa tiến trình.**

* **Lý do:**
  + Trong chương trình của chúng ta sẽ có những lúc chúng ta muốn thực hiện đa tiến trình, những lúc chúng ta muốn thực hiện theo cách thông thường.
* **Giải pháp:**

|  |
| --- |
| class Counter  {      private int Count = 0;      public void increase()      {          Count++;      }      public int getCount()      {          return Count;      }  }  class Main  {      public static void main(String[] args)      {          Counter count = new Counter();          Thread t1 = new Thread( new Runnable()             {                  @Override                  public void run()                  {                      for (int i = 0; i < 1000; i++)                      {                          count.increase();                      }                  }             }          );          Thread t2 = new Thread( new Runnable()             {                  @Override                  public void run()                  {                      for (int i = 0; i < 1000; i++)                      {                          count.increase();                      }                  }             }          );          t1.start();          t2.start();          try {              t1.join();              t2.join();          } catch (InterruptedException e) {              // TODO Auto-generated catch block              e.printStackTrace();          }          System.out.println(count.getCount());      }  } |

* Chú ý rằng đoạn mã join được thêm vào chỉ được phép thêm ở sau t1.start(),.. để chương trình hoạt động đúng cách.

**Đồng bộ đa tiến trình.**

* **Lý do đồng bộ:**
  + Khi các đa tiến trình hoạt động một cách độc lập thì sẽ có lúc trong cùng một thời gian sẽ xảy ra việc hai tiến trình đòi CPU. Điều này sẽ gây ra lỗi.
  + Đồng bộ ra đời để yêu cầu việc đó không xảy ra. Khi có sự bất đồng bộ xảy ra, hệ điều hành sẽ xử lí chúng với sự ưu tiên nhất định.
* **Giải pháp:**
  + Các hàm xử lí trong các tiến trình chạy đa nhiệm cần phải khai báo thêm từ khóa “synchronized”

|  |
| --- |
| class Counter  {      private int Count = 0;      public synchronized void increase()      {          Count++;      }      public int getCount()      {          return Count;      }  }  class Main  {      public static void main(String[] args)      {          Counter count = new Counter();          Thread t1 = new Thread( new Runnable()             {                  @Override                  public void run()                  {                      for (int i = 0; i < 1000; i++)                      {                          count.increase();                      }                  }             }          );          Thread t2 = new Thread( new Runnable()             {                  @Override                  public void run()                  {                      for (int i = 0; i < 1000; i++)                      {                          count.increase();                      }                  }             }          );          t1.start();          t2.start();          try {              t1.join();              t2.join();          } catch (InterruptedException e) {              // TODO Auto-generated catch block              e.printStackTrace();          }          System.out.println(count.getCount());      }  } |

**Project Quản lý thư viện.**

**Đề:**

Thư viện gồm nhiều quyển sách và chúng ta sẽ cho các học sinh đến mượn sách.

|  |
| --- |
| public class Book  {      private String Name;      private String Author;      private boolean Available;      public Book(String Name, String Author)      {          this.Name = Name;          this.Author = Author;          this.Available = true;      }      @Override      public String toString()      {          return Name + " " + Author + " " + Available;      }      public static boolean check\_two\_the\_same\_book(Book b1, Book b2)      {         if (b1.Name.equals(b2.Name) && b1.Author.equals(b2.Author) && b2.Available == true)         {              return true;         }         return false;      }      public void mark\_borrowed\_book()      {          this.Available = !Available;      }  } |

|  |
| --- |
| import java.util.ArrayList;  import java.util.Random;  public class Library  {      private ArrayList<Book> List\_Books = new ArrayList<>();      public void add\_book(Book b)      {          this.List\_Books.add(b);      }      public void display\_books\_in\_Library()      {          for (int i = 0; i < List\_Books.size(); i++)          {              System.out.println(this.List\_Books.get(i));          }      }      public int Random\_Index()      {          Random rd = new Random();          return rd.nextInt(this.List\_Books.size() - 1 - 0 + 1) + 0;      }      public boolean check(Book b)      {          for (int i = 0; i < List\_Books.size(); i++)          {              if (Book.check\_two\_the\_same\_book(b, List\_Books.get(i)) == true)              {                  return true; // có sách.              }          }          return false; // không có sách.      }      public void mark\_borrowed\_book(int Index)      {          this.List\_Books.get(Index).mark\_borrowed\_book();      }      public Book return\_book\_by\_using\_Index(int Index)      {          return this.List\_Books.get(Index);      }  } |

|  |
| --- |
| import java.util.Random;  public class Student implements Runnable  {      private String MSSV;      private Library lb;      private int x = 0;      public Student(String MSSV, Library lb)      {          this.MSSV = MSSV;          this.lb = lb;      }      public synchronized void borrow()      {          Random rd = new Random();          x = this.lb.Random\_Index();          Book b = this.lb.return\_book\_by\_using\_Index(x);          if (this.lb.check(b) == true)          {              System.out.println(MSSV + " da muon sach: " + b);              this.lb.mark\_borrowed\_book(x);              int Borrow\_Time = rd.nextInt(1000 - 0 + 1) + 0;              try {                  Thread.sleep(Borrow\_Time);              } catch (InterruptedException e) {                  // TODO Auto-generated catch block                  e.printStackTrace();              }          }          else          {              System.out.println("Het sach " + b);          }      }      public synchronized void return\_book()      {          Book b = this.lb.return\_book\_by\_using\_Index(x);          System.out.println(MSSV + " Da tra sach: " + b);          this.lb.mark\_borrowed\_book(x);      }      @Override      public void run()      {          while (true)          {              borrow();              return\_book();          }      }  } |

|  |
| --- |
| import java.util.Random;  class Main  {      public static void main(String[] args)      {          Library lb = new Library();          lb.add\_book(new Book("sách 1", "nguyentandung 1" ));          lb.add\_book(new Book("sách 2", "nguyentandung 2" ));          lb.add\_book(new Book("sách 3", "nguyentandung 3" ));          lb.add\_book(new Book("sách 4", "nguyentandung 4" ));          lb.add\_book(new Book("sách 5", "nguyentandung 5" ));          lb.add\_book(new Book("sách 6", "nguyentandung 6" ));          Student st1 = new Student("1", lb);          Student st2 = new Student("2", lb);          Student st3 = new Student("3", lb);          Student st4 = new Student("4", lb);          Student st5 = new Student("5", lb);          Thread th1 = new Thread (st1);          Thread th2 = new Thread (st2);          Thread th3 = new Thread (st3);          Thread th4 = new Thread (st4);          Thread th5 = new Thread (st5);          th1.start();          th2.start();          th3.start();          th4.start();          th5.start();      }  } |