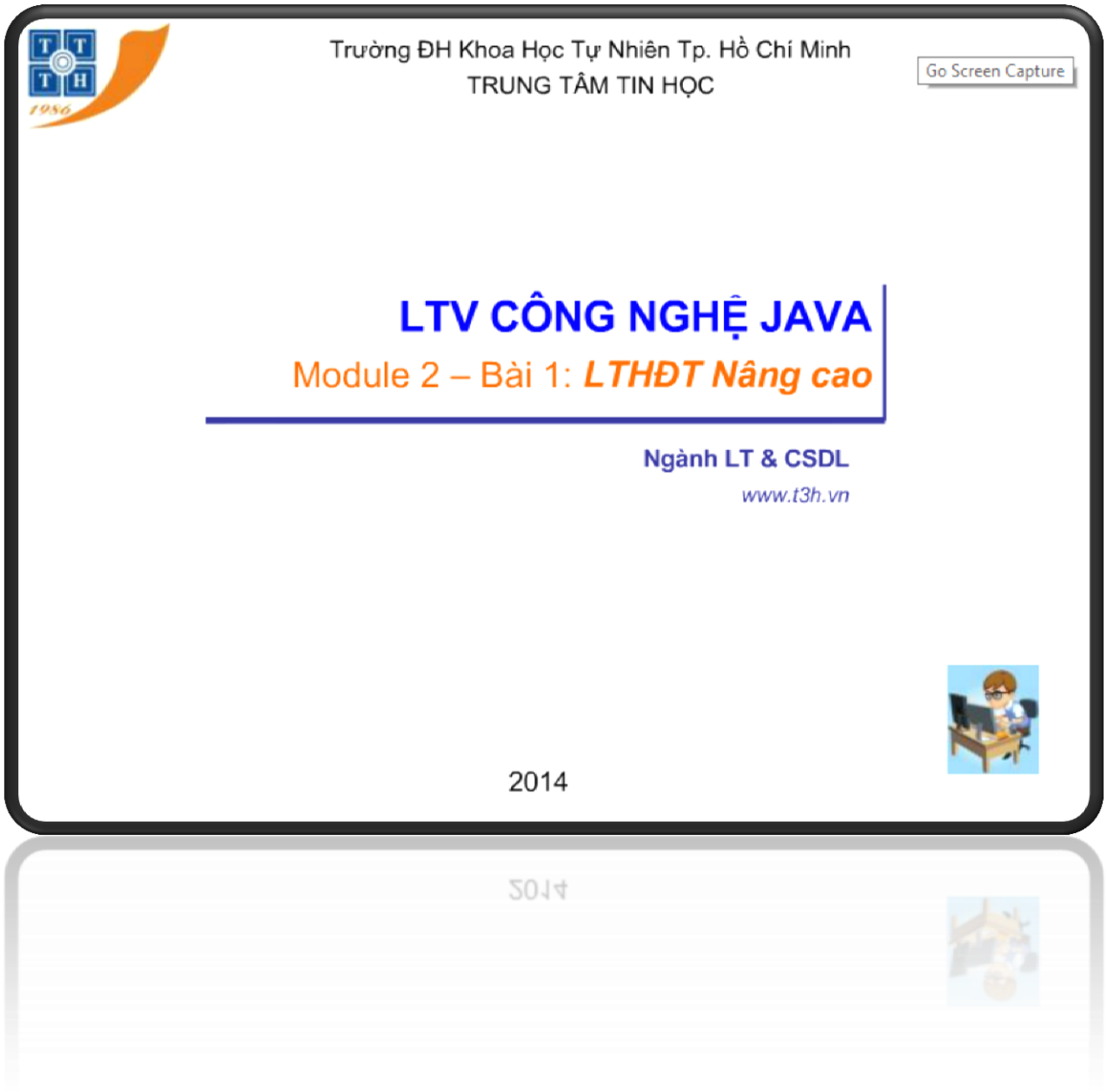
LẬP TRÌNH VIÊN CÔNG NGHỆ JAVA

Module 2

*Click vào phụ lục để chuyển tới **bài cần đọc***

Phụ lục


Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC
Go Screen Capture
LTV CÔNG NGHỆ JAVA
Module 2 – Bài 1: LTHĐT Nâng cao
Ngành LT & CSDL
www.t3h.vn
2014

Nội dung

1. Tính kế thừa và đa hình
2. Abstract (Trừu tượng)
3. Interface (Giao diện)

Tính kế thừa và đa hình

❑Khái niệm

- Kế thừa (Inheritance): Quá trình mà các tính chất và hành vi được truyền từ thực thể cha đến thực thể con
- Kế thừa giúp tránh phải tạo các tính chất và hành vi đã có sẵn và sử dụng lại cái đã có sẵn để tạo ra thực thể mới

Tính kế thừa và đa hình

Tính năng và thuật ngữ

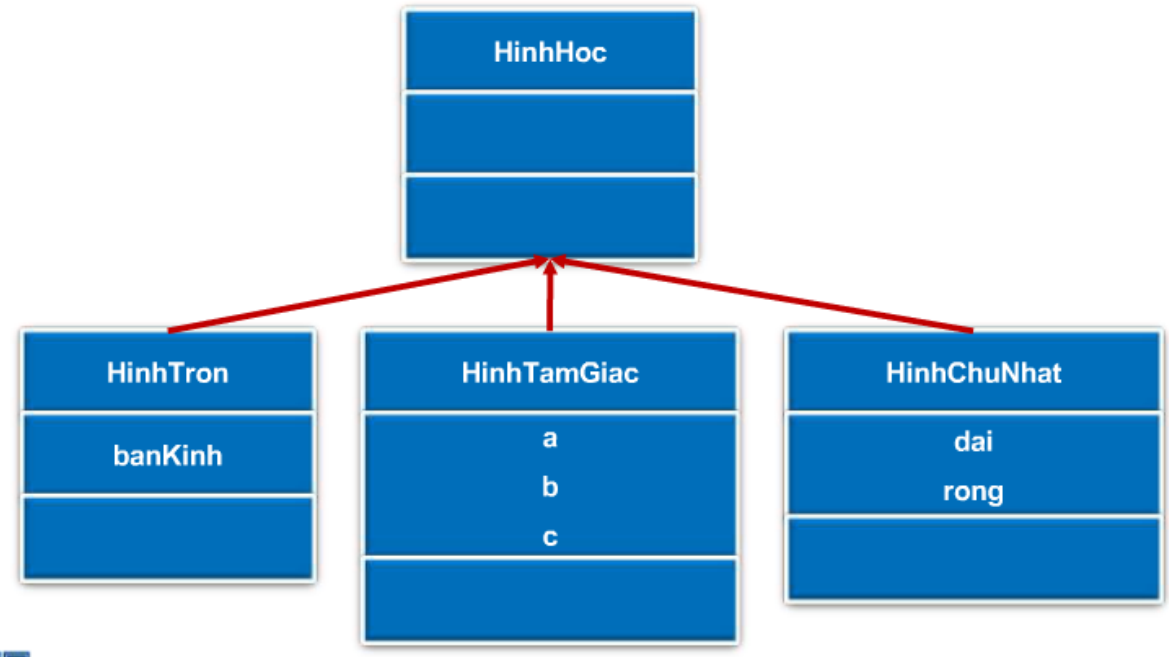
- Kế thừa cho phép định nghĩa một **lớp tổng quát** trước, và định nghĩa các **lớp chuyên biệt** sau bằng cách cài đặt thêm các chi tiết vào định nghĩa lớp
- Lớp mà kế thừa các phương thức và các thuộc tính gọi là **lớp con** (lớp dẫn xuất)
- Lớp mà cho phép lớp con kế thừa các phương thức và các thuộc tính gọi là **lớp cha** (lớp cơ sở)

Tính kế thừa và đa hình

Lớp con

- Sử dụng từ khóa **extends** để tạo ra lớp con
- Một lớp con chỉ có thể kế thừa trực tiếp từ một lớp cha
- Nếu lớp con không kế thừa từ lớp cha nào hết, mặc định xem như nó kế thừa từ lớp cha tên là **Object**
- Hàm dựng không được kế thừa.

Tính kế thừa và đa hình



Tính kế thừa và đa hình

❑Khái niệm kế thừa đa hình

```
class HinhHoc{
   public HinhHoc(){
       System.out.println("Hinh hoc khong tham so");
   }
   public HinhHoc(int giaTri){
       System.out.println("Hinh hoc co tham so");

   }
}
```


Tính kế thừa và đa hình

❑Khái niệm kế thừa đa hình

```
class HinhHoc{
   public HinhHoc(){
        System.out.println("Hinh hoc khong tham so");
   }
   public HinhHoc(int giaTri){
        System.out.println("Hinh hoc co tham so");
   }
}
```

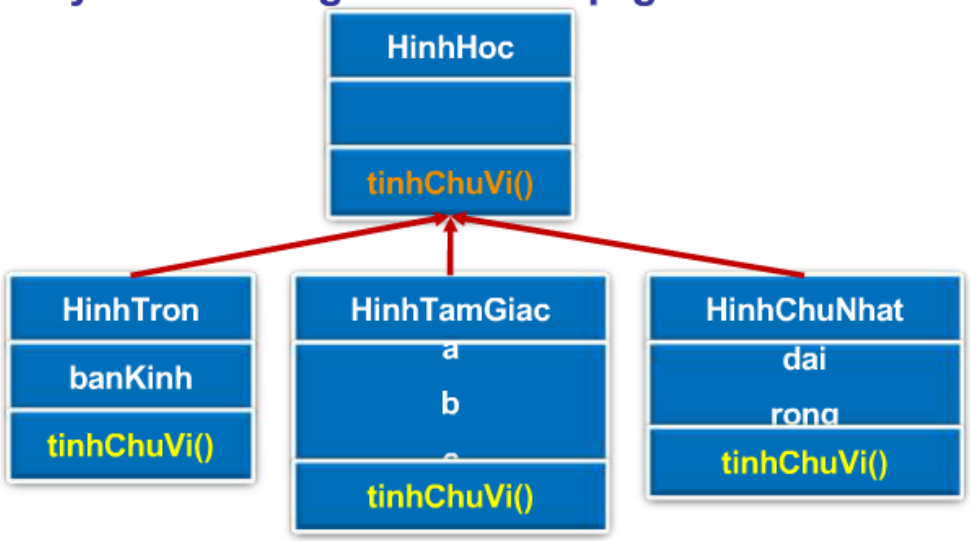
Tính kế thừa và đa hình

❑Khái niệm kế thừa đa hình

```
class HinhTron extends HinhHoc{
   double bankinh;
   public HinhTron(){
        System.out.println("Hinh tron");
   }
}
class HinhTamGiac extends HinhHoc{
   double a,b,c;
   public HinhTamGiac(int a){
        super(a); System.out.println("Hinh tam giac");
   }
}
```

Tính kế thừa và đa hình

- Tính đa hình (polymorphism): một thực thể có nhiều hình thức, trạng thái khác nhau tùy thuộc vào yêu cầu và ngữ cảnh sử dụng



Tính kế thừa và đa hình

- Ghi đè hàm (overriding)
  - Lớp con định nghĩa một phương thức mới trùng tên và trùng tham số với một phương thức của lớp cha
  - Phương thức ghi đè ở lớp con phải:
    - Cùng tên và tham số
    - Không được sử dụng chỉ định từ truy xuất yếu hơn
  - Từ khóa **super**
    - Truy xuất đến phương thức cùng tên của lớp cha
    - Truy xuất đến hàm dựng của lớp cha

Tính kế thừa và đa hình

- **Nạp chồng hàm**
  - Là khả năng một lớp có nhiều phương thức có cùng tên
    - Khác nhau về số tham số và có cùng kiểu dữ liệu của tham số
    - Có cùng số tham số và khác nhau kiểu dữ liệu của tham số
    - Hàm dựng cũng có khả năng nạp chồng với quy tắc như trên

Tính kế thừa và đa hình

- **Nạp chồng hàm**
  - Ví dụ nạp chồng hàm

```
class HinhChuNhat extends HinhHoc{
  double dai,rong;
  public HinhChuNhat(){}
  public HinhChuNhat(double d,double r){
     dai=d;rong=r;
  }
  public HinhChuNhat(double a) {
  dai=rong=a; }
}
```

Tính kế thừa và đa hình

- **super**
  - Được dùng trong **hàm dựng** hoặc trong **hàm thực thể** để chỉ đến đối tượng cha của đối tượng hiện hành
  - Được dùng như một **thể hiện của lớp cha** trong khai báo của lớp con

Tính kế thừa và đa hình

- **super**
  - Ví dụ:

```
class HinhChuNhat extends HinhHoc{
  double dai,rong;
  public HinhChuNhat(){}
  public HinhChuNhat(double dai,double rong){
      super();
      this.dai = dai;this.rong = rong;
  }
  public HinhChuNhat(double a) { dai = rong = a; }
}
```

Tính kế thừa và đa hình

- this
  - Được dùng trong **hàm dựng** hoặc trong **hàm thực thể** để **chỉ đến đối tượng hiện hành**
  - Được dùng để truy xuất đến các biến thành viên mà bị các biến cục bộ che khuất vì khai báo trùng tên

Tính kế thừa và đa hình

- this
  - Ví dụ

```
class HinhChuNhat extends HinhHoc{
  double dai,rong;
  public HinhChuNhat(){}
  public HinhChuNhat(double dai,double rong){
      this.dai=dai;this.rong=rong;
  }
  public HinhChuNhat(double a) { dai=rong=a; }
}
```

Tính kế thừa và đa hình

- **static**
  - Biến thuộc lớp
    - Được khai báo sử dụng từ khóa **static**
    - Khi giá trị của biến thay đổi, tất cả các đối tượng của lớp sẽ được cập nhật để nhìn thấy giá trị mới của biến
    - *Có thể được quản lý mà không cần phải tạo ra thực thể của lớp*
    - *Được truy xuất tới thông qua tên lớp*

Tính kế thừa và đa hình

- **static**
  - Ví dụ

```
class MyClass{
  public static int
  giatri=10;
  public void Tang(int
  x){
      giatri+=x;
    }
  public void Xuat(){

  System.out.println(gia
  tri);
  }
```

```
public static void
  main(String args[]){
  MyClass a=new
  MyClass();
  MyClass b=new Class();
  System.out.println(
  MyClass.giatri);
  a.Tang(10); b.Xuat();
}
```

Tính kế thừa và đa hình

❑Hàm static

- Cũng được xem là phương thức của lớp, nhưng không tham chiếu tới bất kỳ thực thể nào của lớp, thường dùng để truy xuất tới các biến và hàm thành viên
- Từ khóa **this** không được dùng bên trong hàm static
- *Có thể được truy xuất bằng cách sử dụng tên lớp và tên thực thể*

Tính kế thừa và đa hình

❑Hàm static

```
class MyClass{
   public static int giatri
   = 10;
   public void Tang(int x){
       giatri += x;
     }
   public void Xuat(){

   System.out.println(giatri
   );
   }
```

```
public static void
   main(String args[]){
   MyClass a = new
   MyClass();
   MyClass b = new
   Class();
   System.out.println(
   MyClass.giatri);
   a.Tang(10);
   b.Xuat();
}
```

Tính kế thừa và đa hình

❑Khối lệnh static

- Là khối lệnh nằm trong cặp dấu ngoặc nhọn
- Bắt đầu bằng từ khóa static
- *Dùng để khởi tạo các biến static của lớp*
- Khối lệnh khởi tạo static chỉ dùng để tham chiếu đến các biến trong lớp mà đã được khai báo trước đó

Tính kế thừa và đa hình

❑static

```
class MyClass{
   static int counter = 0;
   static {
      counter++;
      System.out.println("Khoi
lenh static: counter=" +
counter);
   }
MyClass(){System.out.println("
Ham khoi tao: counter=" +
counter);}
}
```

```
public static void
main(String args[]){
      MyClass m = new
MyClass();
      System.out.println("
Phuong thuc main: counter =
"+MyClass.counter);
      //System.out.println
("Phuong thuc main: counter
= "+m.counter);
}
```

Nội dung

1. Tính kế thừa và đa hình
2. Abstract (Trừu tượng)
3. Interface (Giao diện)

Abstract (Trừu tượng)

- Tính trừu tượng được sử dụng trong các tình huống đối tượng có tồn tại nhưng thực sự nó thuộc một đối tượng con cụ thể nào đó

- Đặc tả abstract áp dụng cho:
  - lớp gọi là lớp trừu tượng (abstract class)
  - hàm gọi là hàm trừu tượng (abstract method)

```
abstract class <Tên Lớp> {}
abstract <Kiểu dữ liệu> <Tên hàm>(<Đối số>);
```

Chỉ là khai báo, không hiện thực

Abstract (Trừu tượng)

- Lớp trừu tượng không khởi tạo thực thể vì chưa hiện thực đầy đủ
- Lớp trừu tượng chứa một hoặc nhiều hàm trừu tượng. Tuy nhiên, **không** có hàm trừu tượng vẫn khai báo lớp trừu tượng
- Một lớp chứa hàm trừu tượng thì lớp đó phải là lớp trừu tượng
- Một lớp thừa kế lớp trừu tượng phải hiện thực tất cả các hàm trừu tượng hoặc lớp đó cũng là lớp trừu tượng

Abstract (Trừu tượng)

- Lớp trừu tượng hiện thực (implements) một giao diện (interface) và có thể không hiện thực một hàm

```
abstract class Shape {
   abstract void draw();
   void message() {
       System.out.println("I cannot live without being a
parent.");
   }
}
class Circle extends Shape {
   void draw() {
       System.out.println("Circle drawn.");
   }
}
```

Abstract (Trừu tượng)

```
class Cone extends Shape {
   void draw() {
        System.out.println("Cone drawn.");
   }
}
public class RunShape {
   public static void main(String[] args) {
        Circle circ = new Circle();
        Cone cone = new Cone();
        circ.draw();
        cone.draw();
        cone.message();
   }
}
```

Nội dung

1. Tính kế thừa và đa hình
2. Abstract (Trừu tượng)
3. Interface (Giao diện)

Interface (Giao diện)

- Thể hiện tính bao đóng (encapsulation) trong lập trình hướng đối tượng
- Hỗ trợ đa thừa kế: một lớp có thể thừa kế một lớp và hiện thực nhiều giao diện

  <subclass> extends <superclass> implements <interface1>, <interface2>{}

- Lớp hiện thực giao diện phải hiện thực tất cả các hàm được đặc tả trong giao diện đó

Interface (Giao diện)

- Cấu trúc định nghĩa:

```
interface <InterfaceName> {
    <dataType1> <var1>;
    <dataType2> <var2>;
    <ReturnType1> <methodName1> ( );
    <ReturnType2> <methodName2>(<parameters>);
}
```

- Hàm khai báo có đặc tả ẩn public và abstract
- Biến khai báo có đặc tả ẩn là public, static và final

Interface (Giao diện)

```
interface ParentOne {
    int pOne = 1;
    void printParentOne();
}
interface ParentTwo {
    int pTwo = 2;
    void printParentTwo();
}
interface Child extends ParentOne,
ParentTwo{
     int child = 3;
     void printChild();
}
```


Interface (Giao diện)

```
class InheritClass implements
Child {
    public void printParentOne(){
        System.out.println(pOne);
    }
    public void printParentTwo(){
        System.out.println(pTwo);
    }
    public void printChild(){

System.out.println(child);
    }
}
```

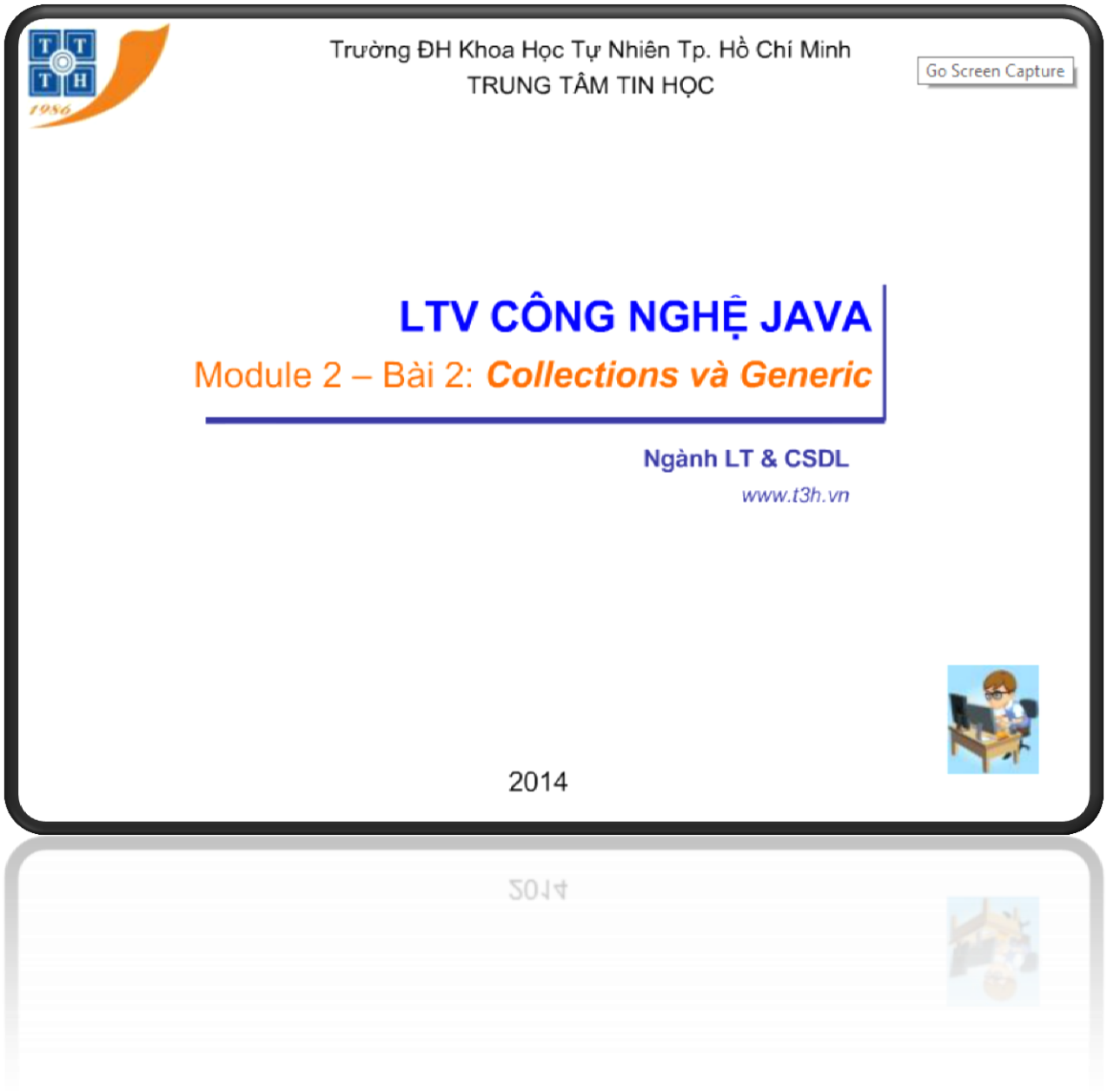
```
class TestInterface {
    public static void
main(String[] args){
        InheritClass ic
= new InheritClass();

ic.printParentOne();

ic.printParentTwo();
        ic.printChild();
    }
}
```



LTV Công nghệ Java – Module 2
34


Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC
Go Screen Capture
LTV CÔNG NGHỆ JAVA
Module 2 – Bài 2: Collections và Generic
Ngành LT & CSDL
www.t3h.vn
2014