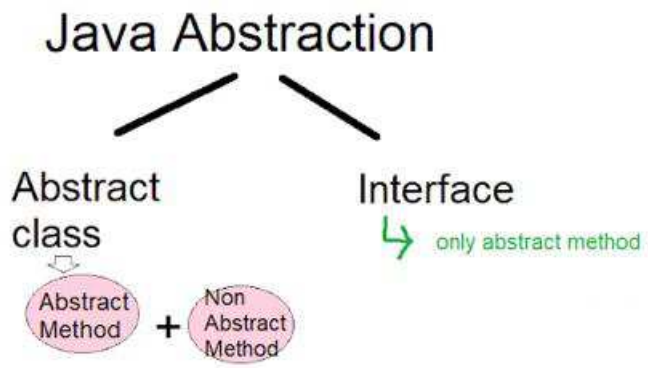
Resource Software Solution

Session 08:
Polymorphism & Abstraction



Nội dung

1. Tính đa hình (Polymorphism)
2. Tính trừu tượng (Abstraction)
3. Từ khóa static

Tính đa hình (1)

Polymorphism

- Polymorphism có nghĩa là "many forms – **nhiều hình thức**" và nó xảy ra khi có **nhiều lớp có liên quan với nhau thông qua tính kế thừa**
- Tính đa hình cho phép chúng ta thực hiện **một hành động** theo những **cách khác nhau**

Tính đa hình (2)

Ví dụ 1

```
public class Calculator {
	public float sum(float n1, float n2) {
		return n1 + n2;
	}

	public int sum(int n1, int n2) {
		return n1 + n2;
	}
}
```

Thực hiện **một hành động tính tổng** theo những **cách khác nhau**

```
Calculator c = new Calculator();
int result1 = c.sum(10, 20);
float result2 = c.sum(10.2, 20.8);

System.out.println(result1 + ","+ resulat2);
```

Tính đa hình (3)

Ví dụ 2

- Khai báo lớp Person

```
public Person {
  protected String name;
  protected String address;

  public Person (String name, String address) {
     this.name = name;
     this.address = address;
  }

  protected void display() {
      System.out.println("Name: " + name +", address: " + address);
   }
}
```

Tính đa hình (4)

Ví dụ 2

- Khai báo lớp **Programmer** kế thừa lớp **Person**

```
Public Programmer extends Person {
  private String companyName;

  public Programmer (String name, String address, String companyName) {
    super(name, address);
    this.companyName = companyName;
  }

  @Override
  protected void display() {
     super.display();
     System.out.println("Company name: " + companyName);
  }
}
```

Tính đa hình (5)

Ví dụ 2

- Khai báo lớp thực thi **EmployeeManagement**

```
public EmployeeManagement {
   public static void main(String[] args) {
      Person p = new Programmer("Lê Hồng Kỳ", "HCM", "R2S");
      p.display();
   }
}
```

Nhiều hình thức khi có **nhiều lớp có liên quan với nhau thông qua tính kế thừa**

Tính đa hình (6)

Overloading vs Overriding

Overloading	Overriding
Cơ chế cho phép khai báo các phương thức **trùng tên** nhưng **khác tham số** • **Số lượng** tham số khác nhau • Hoặc cùng số lượng nhưng khác **kiểu dữ liệu** • Hoặc cùng số lượng, cùng kiểu dữ liệu nhưng khác nhau về **thứ tự**	Cơ chế cho phép lớp con thay đổi phần thân phương thức của lớp cha tại lớp con • **Phần khai báo** giống nhau giữa lớp cha và lớp con • **Phần thân** được định nghĩa lại tại lớp con

Resource Software Solution

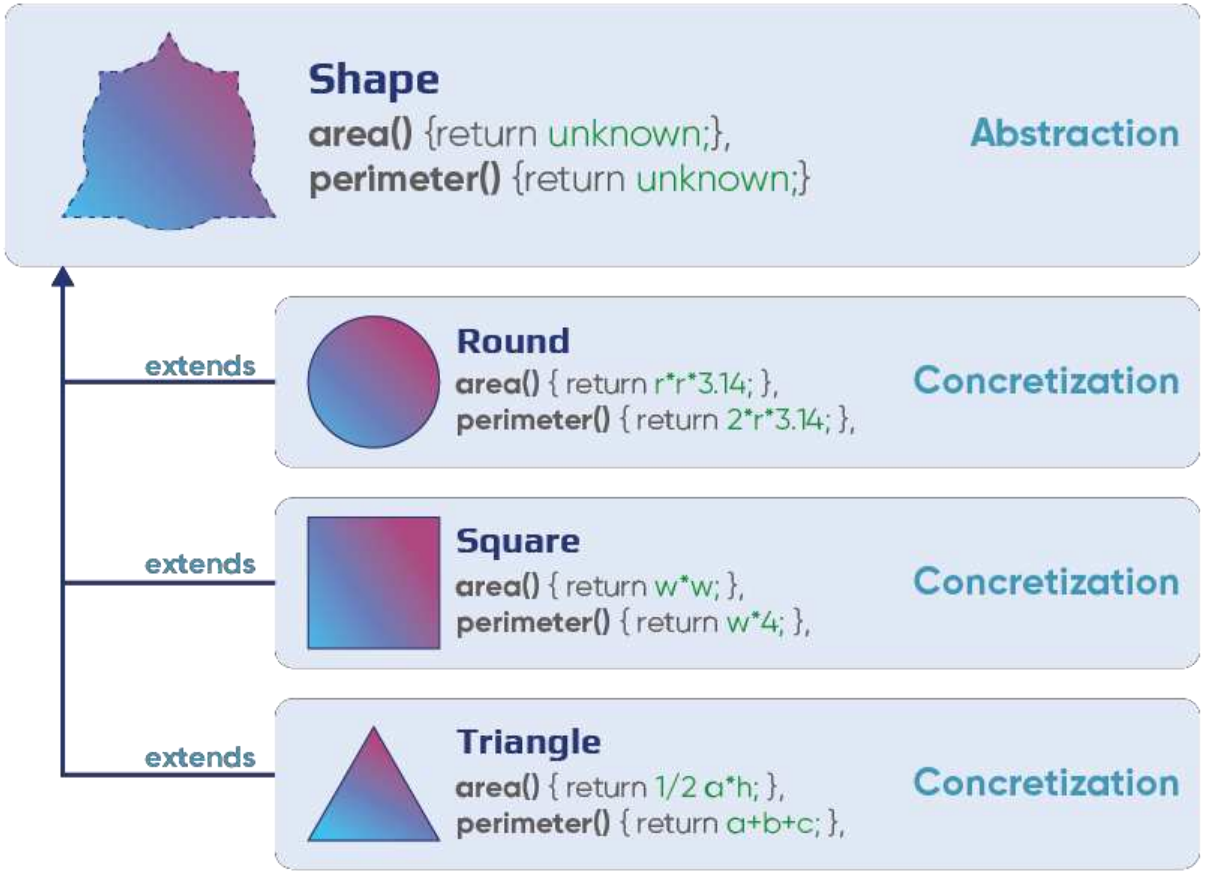
2

Tính trừu tượng (Abstraction)

Tính trừu tượng (1)

Abstraction

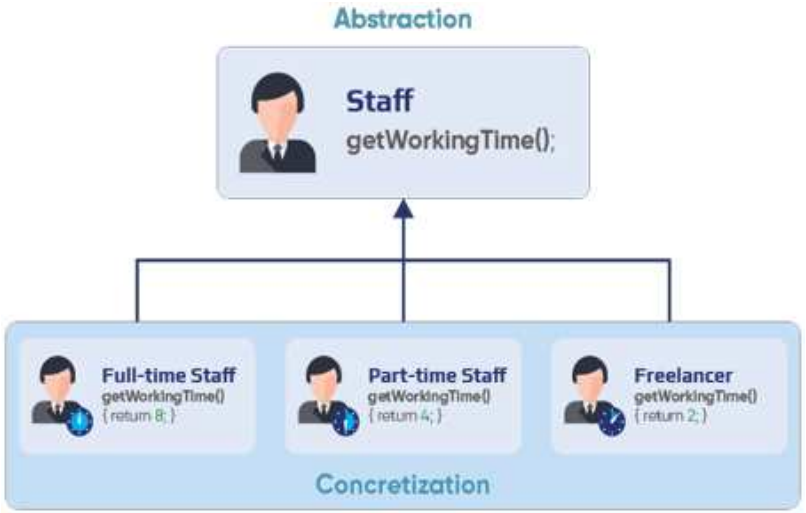
- Trong lập trình, khái niệm **trừu tượng** được xem như là một khuông mẫu không có hình thù xác định (chỉ có khung sườn chứa **những hành vi chưa được làm rõ**)
- Các **thành viên** hiện thực sự trừu tượng bằng cách **cụ thể hoá** các hành vi đó



Tính trừu tượng (2)

Ví dụ

- Tất cả **nhân viên** trong công ty đều được quy định về thời gian làm việc
- **Nhân viên toàn thời gian** làm 8 tiếng/ngày
- **Nhân viên bán gian** làm 4 tiếng/ngày
- **Cộng tác viên** làm 2 tiếng/ngày

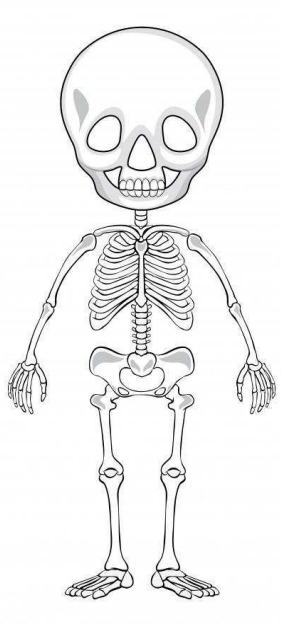


- **Nhân viên** được xem là **trừu tượng**
- Nhân viên toàn thời gian, bán thời gian, cộng tác viên là **cụ thể hoá từ Nhân viên**

Tính trừu tượng (3)

Hình thức thể hiện

- Có hai hình thức thể hiện
  1. **Interface**
  2. **Abstract class**



Interface

Abstract class

Tính trừu tượng (4)

Interface

- Một giao diện (interface) chỉ khai báo các **phương thức** của đối tượng
  - Xác định loại **hoạt động** mà một đối tượng có thể thực hiện
  - Phương thức này chưa có phần thân (trừu tượng)
- Các hoạt động này được định nghĩa bởi các lớp triển khai giao diện bằng từ khóa "**implements**"
- Không thể tạo đối tượng từ Interface vì mọi thứ vẫn còn khái quát (Chỉ chứa những **phương thức không có nội dung**)

Tính trừu tượng (5)

Interface – Ví dụ

```
public interface Actionable {
    public abstract void input();
    public abstract void output();
}

public class Customer implements Actionable {
    @Override
    public void input() {
    }

    @Override
    public void output() {
    }
}
```

Khai báo phương thức không có phần thân

Cụ thể hóa xử lý

Tính trừu tượng (6)

Abstract class

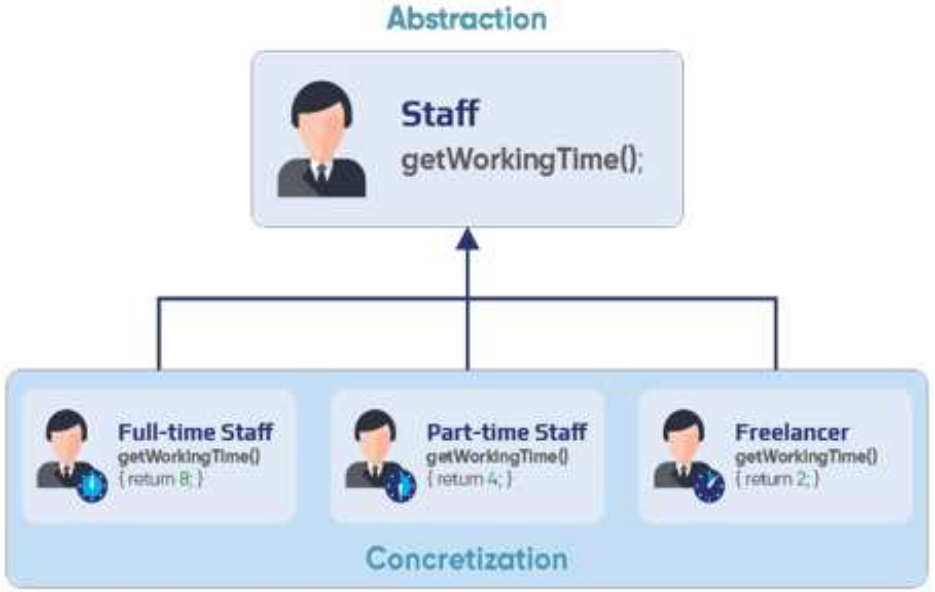
- Lớp trừu tượng cũng là **một lớp** (class), nghĩa là cũng có **thuộc tính** và **phương thức** như một lớp bình thường
- Ngoài ra nó có thể **bao gồm** hoặc **không bao gồm** các phương thức trừu tượng
- Lớp con thường
  - Cung cấp **các triển khai** cho tất cả các phương thức trừu tượng
  - Hoặc **lớp con** cũng phải được khai báo là trừu tượng
- **Không** thể tạo đối tượng từ lớp trừu tượng

Tính trừu tượng (7)

Abstract class – Ví dụ 1

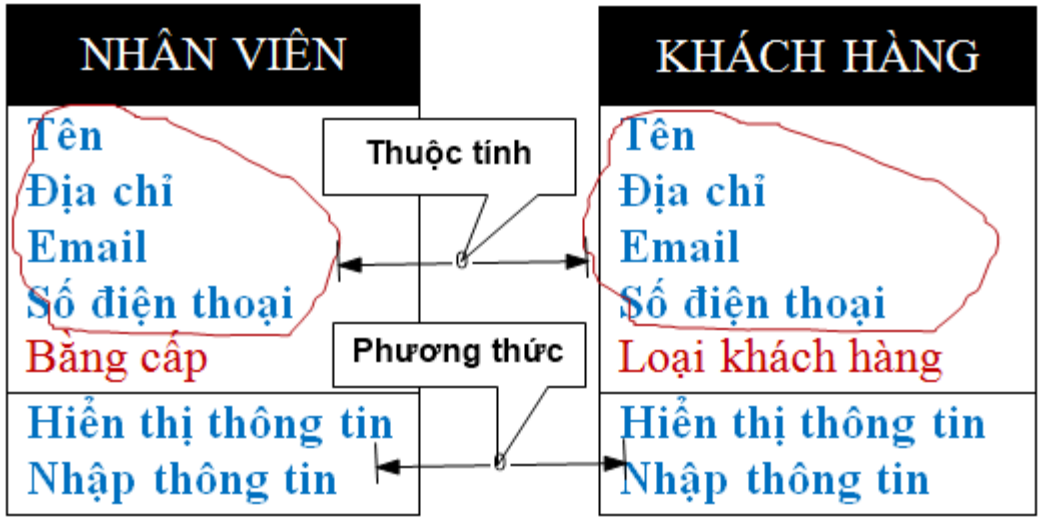
```
public abstract class Staff {
    public abstract byte getWorkingTime();
}

public class FulltimeStaff extends Staff {
   @Override
    public byte getWorkingTime() {
          return 8;
    }
}
```

Chứa phương thức trừu tượng



Tính trừu tượng (8)

Abstract class – Ví dụ 2

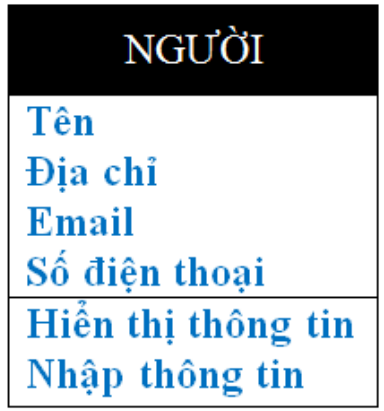


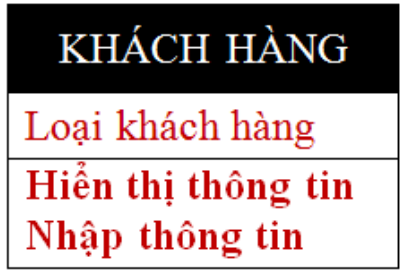
Tính trừu tượng (9)

Abstract class – Ví dụ 2

```
public abstract class Nguoi {
    protected String ten;

    protected void hienThi() {

    }
}

public class NhanVien extends Nguoi {
    private String bangCap;
}
```

Không chứa phương thức trừu tượng





Tính trừu tượng (10)

Abstract class vs Interface

Abstract class	Interface
Lớp trừu tượng có thể có các phương thức trừu tượng hoặc không	Chỉ có thể có các phương thức trừu tượng (Từ Java 8, có thể có các phương thức mặc định và tĩnh)
Không hỗ trợ đa thừa kế	Có hỗ trợ đã thừa kế
Chứa các biến là final hoặc non-final, static hoặc non-static	Chỉ có static và final (Hằng số)
Có thể kế thừa và implement nhiều interface	Chỉ có thể kế thừa interface
Có thể sử dụng private, protected, …	Các thành viên mặc định là public
public abstract class Shape { ** public abstract void draw();** **}**	**public interface Drawable {** ** void draw();** **}**

Resource Software Solution

3

Từ khóa static

Từ khóa static trong java (1)

Giới thiệu

- Từ khóa **static** được sử dụng để quản lý bộ nhớ tốt hơn và nó có thể được truy cập **trực tiếp** thông qua lớp mà **không cần** khởi tạo
- Thành viên static **thuộc về lớp** chứ không thuộc về thể hiện/Đối tượng (instance) của lớp
- Lập trình viên có thể áp dụng từ khóa static với
    1. Các **biến**
    2. Các **phương thức**
    3. Các **lớp lồng nhau** (nested class)

Từ khóa static trong java (2)

Biến với static

- Việc cấp phát bộ nhớ cho biến static chỉ xảy ra **một lần** khi class được nạp vào bộ nhớ
- Biến static có thể được sử dụng làm **thuộc tính chung** cho **tất cả đối tượng** của lớp đó
- Nếu một biến vừa khai báo từ khóa **final** vừa khai báo từ khóa **static** thì nó được xem như là một **hằng số**

Từ khóa static trong java (3)

Biến với static – Ví dụ

```
public class Common {
    public static final String COMPANY_NAME = "R2S";
}

public class Program {
  public void display() {
        System.out.println("Company name: " + Common.COMPANY_NAME);
  }
}
```

Truy cập trực tiếp qua lớp

Từ khóa static trong java (4)

Biến với static – Ví dụ

```
public class Counter {
    int count = 0;

    public void visit() {
        count++;
        this.print();
    }

    public void print() {
        System.out.println("count = " + count);
    }
```

```
public static void main(String[] args) {
        Counter c1 = new Counter();
        c1.visit();

        Counter c2 = new Counter();
        c2.visit();
    }
}
```

Output:

```
count = 1
count = 1
```

Từ khóa static trong java (5)

Biến với static – Ví dụ

```
public class Counter {
   static int count = 0;

   public void visit() {
      count++;
      this.print();
   }

   public void print() {
      System.out.println("count = " + count);
   }
```

Dùng chung cho tất cả các đối tượng

```
public static void main(String[] args) {
      Counter c1 = new Counter();
      c1.visit();

      Counter c2 = new Counter();
      c2.visit();
   }
}
```

Output:

```
count = 1
count = 2
```

Từ khóa static trong java (6)

Phương thức với static

- Khi phương thức **không** phụ thuộc vào trạng thái của **đối tượng**
  - Không cần sử dụng bất kỳ dữ liệu thành viên nào của đối tượng
  - Dữ liệu được truyền như các tham số (parameters)
- Các phương thức **tiện ích** là một trường hợp thường được sử dụng nhất trong Java
- Sử dụng trong các **Design Pattern** cần truy cập global như
  - Singleton pattern
  - Factory pattern
  - …

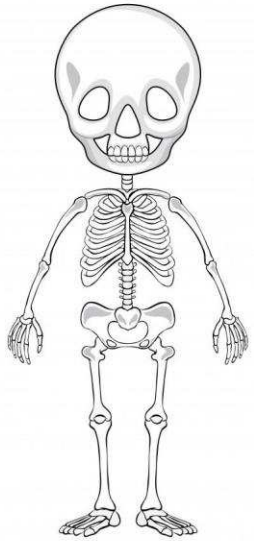
Từ khóa static trong java (7)

Phương thức với static – Ví dụ

```
public class Common {
    public static String companyName = "R2S";

   public static void change (String companyName) {
        this.companyName = companyName;
   }
}

public class Program {
   public void display() {
        System.out.println(Common.change("R2S Corp"));
   }
}
```

Tổng kết

- Tính đa hình
  - Overloading
  - Overriding
- Tính trừu tượng
  - Interface
  - Abstract class
- Từ khóa static



Thankyou!