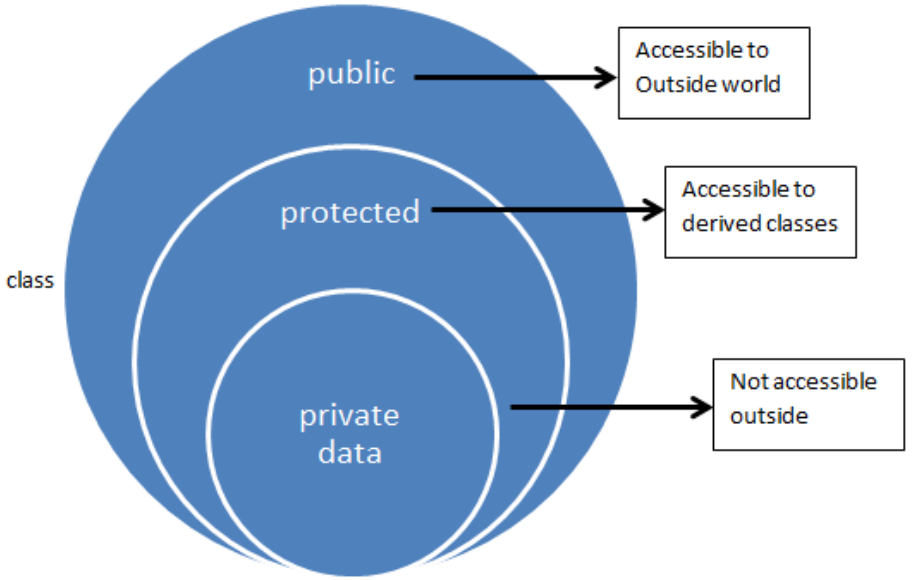
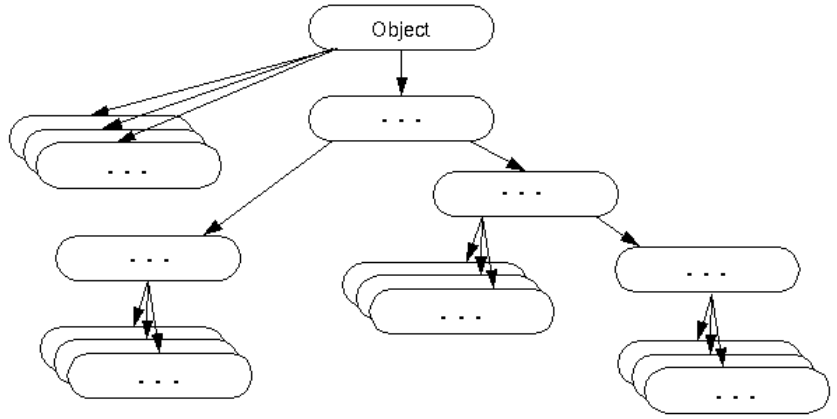
Resource Software Solution

Session 07:
Encapsulation & Inheritance





Nội dung

1. Tính đóng gói (Encapsulation)
2. Tính kế thừa (Inheritance)
3. Từ khóa final
4. Lớp Object

Cho một ví dụ

Non-Encapsulation

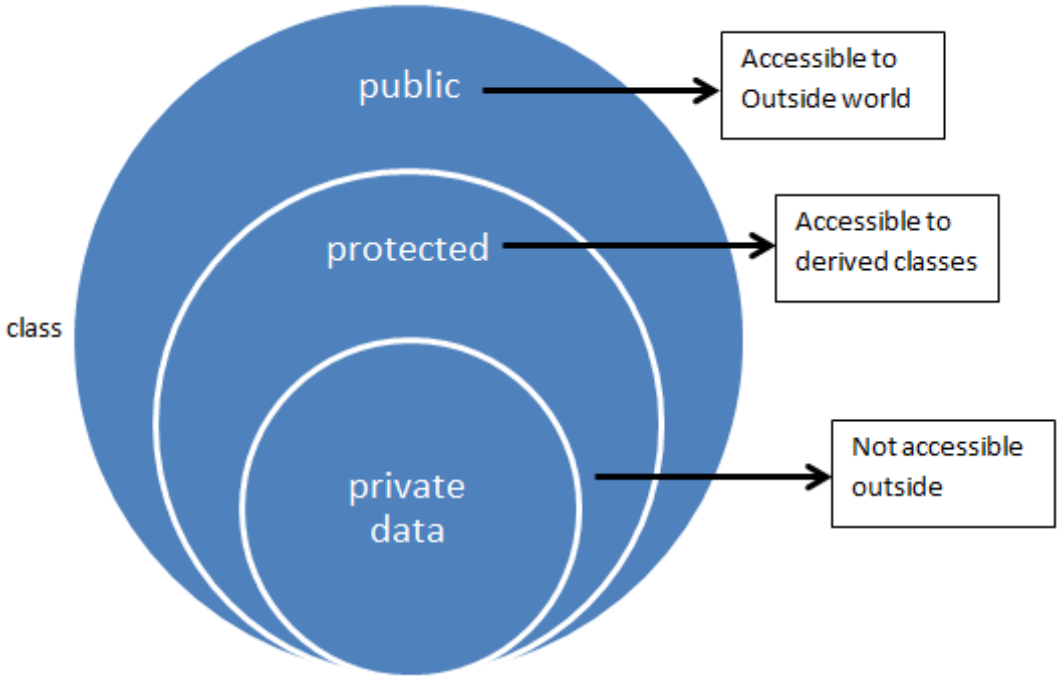
- Giả sử định nghĩa lớp **SinhVien** và **công khai** thuộc tính họ tên và điểm
- Khi sử dụng, người dùng có thể **gán dữ liệu** cho các thuộc tính một cách **tuỳ tiện**

```
public class SinhVien {
  public String hoTen;
  public float diem;
}

public class QuanLySinhVien {
  public stattic void main(String[] args) {
    SinhVien sv = new SinhVien();
    sv.hoTen = "Lê Hồng Kỳ";
    sv.diem = 15.5;
  }
}
```

Chuyện gì đang xảy ra

Tính đóng gói (1)
Encapsulation

- Mọi **dữ liệu (thuộc tính)** của lớp nên được **che dấu** từ bên ngoài
- **Tính đóng gói** sử dụng **phương thức để truy xuất** các thuộc tính được che dấu này
- Mục đích của sự che dấu là để
  - **Bảo vệ dữ liệu**
  - **Tăng cường khả năng mở rộng**

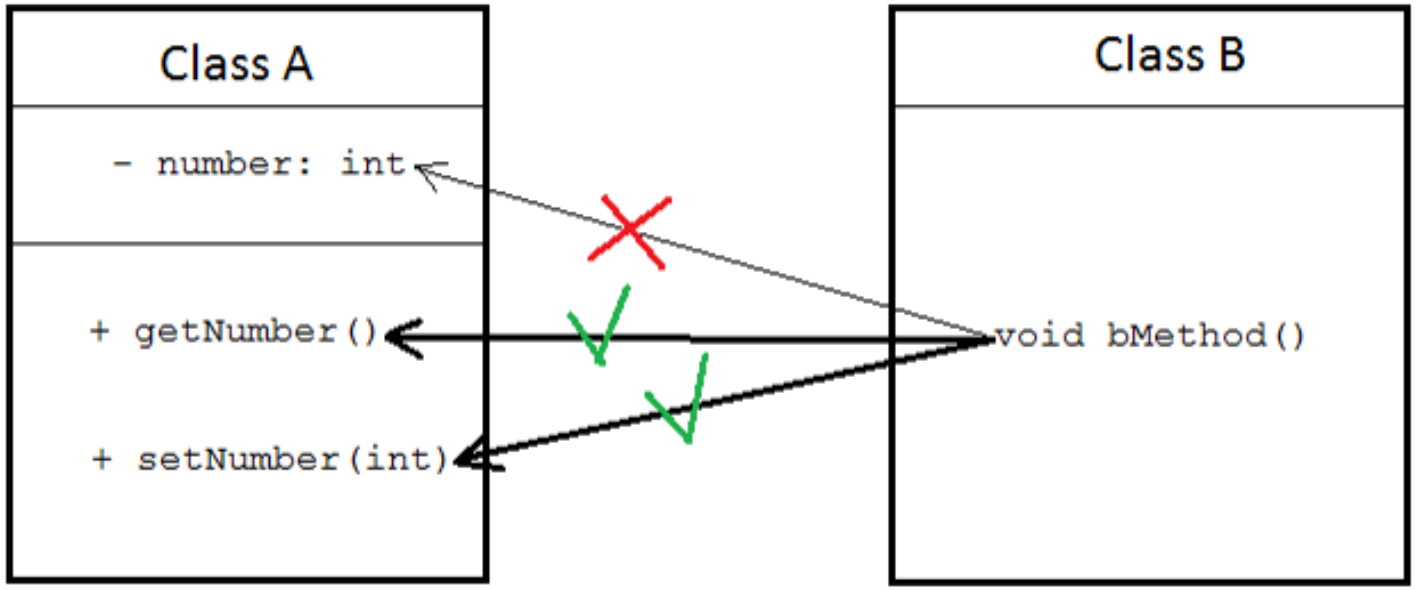


Tính đóng gói (2)
Encapsulation

- Để che dấu dữ liệu, lập trình viên sử dụng **private** khi khai báo các thuộc tính

```
public class SinhVien {
   private String hoTen;
   private float diem;
}
```

Che dấu thông tin



Tính đóng gói (3)
Encapsulation

- Bổ sung các phương thức **public**
  - **getter: Lấy dữ liệu** của các thuộc tính đã che dấu. Thuộc tính chỉ đọc, không được phép chỉnh sửa dữ liệu
  - **setter: Thiết lập dữ liệu** cho các thuộc tính đã che dấu. Thuộc tính chỉ ghi, không được phép hiển thị dữ liệu

```
public class SinhVien {
  private String hoTen;
  private float diem;
```

```
  public float getDiem() {
     return diem;
  }
```

```
  public void setDiem(float diem) {
     if (diem >=0 && diem <=10) {
        this.diem = diem;
     }
  }
}
```

Tính đóng gói (4)

Example

```
public class SinhVien {
   private String hoTen;
   private float diem;

   public float getDiem() {
      return diem;
   }

  public void setDiem(float diem) {
      if (diem >=0 && diem <=10) {
         this.diem = diem;
      }
   }
}
```

```
public class QuanLySinhVien {
   public stattic void main(String[] args) {
      SinhVien sv = new SinhVien();
      sv.diem = 15.5;
   }
}
```

Báo lỗi vì phạm vi private

```
public class QuanLySinhVien {
   public stattic void main(String[] args) {
      SinhVien sv = new SinhVien();
      sv.setDiem(15.5);
   }
}
```

Không thay đổi được

Tính đóng gói (5)

Tổng kết

- Mục đích của tính đóng gói là **bảo vệ dữ liệu** và **tăng cường khả năng mở rộng**
- Đối với những thuộc tính muốn sử dụng **tính đóng gói thì khai báo với quyền truy cập là private**
- Sử dụng **getter** và **setter** để giúp **truy cập những thuộc tính là private**

```
public class Person {
 // private = restricted access
  private String name;

  // Getter
  public String getName() {
    return name;
  }

  // Setter
  public void setName(String newName) {
    this.name = newName;
  }
}
```

Resource Software Solution

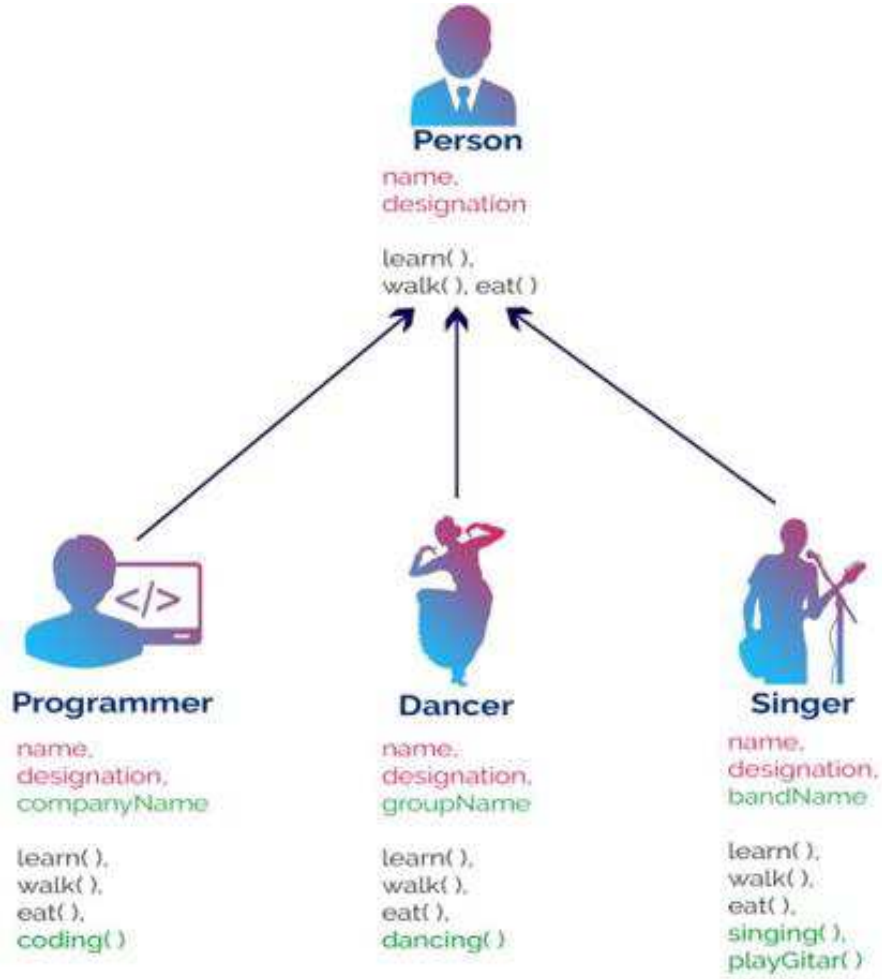
2

Tính kế thừa (Inheritance)

Tính kế thừa (1)

Sự phân cấp

- Các lớp tồn tại trong một hệ thống thứ bậc phân cấp gọi là cây thừa kế
- Lớp bậc trên gọi là **lớp cha (super class)**
- Lớp bậc dưới gọi là **lớp con (sub class)**
- Trong Java một lớp con chỉ có một lớp cha duy nhất (**đơn thừa kế**)

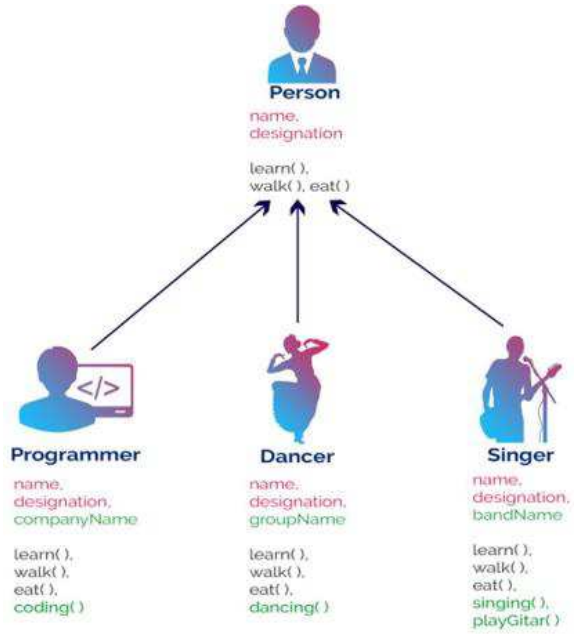


Tính kế thừa (2)

Thể hiện kế thừa

extends dùng để chỉ định lớp con (sub class) kế thừa từ lớp cha (super class)

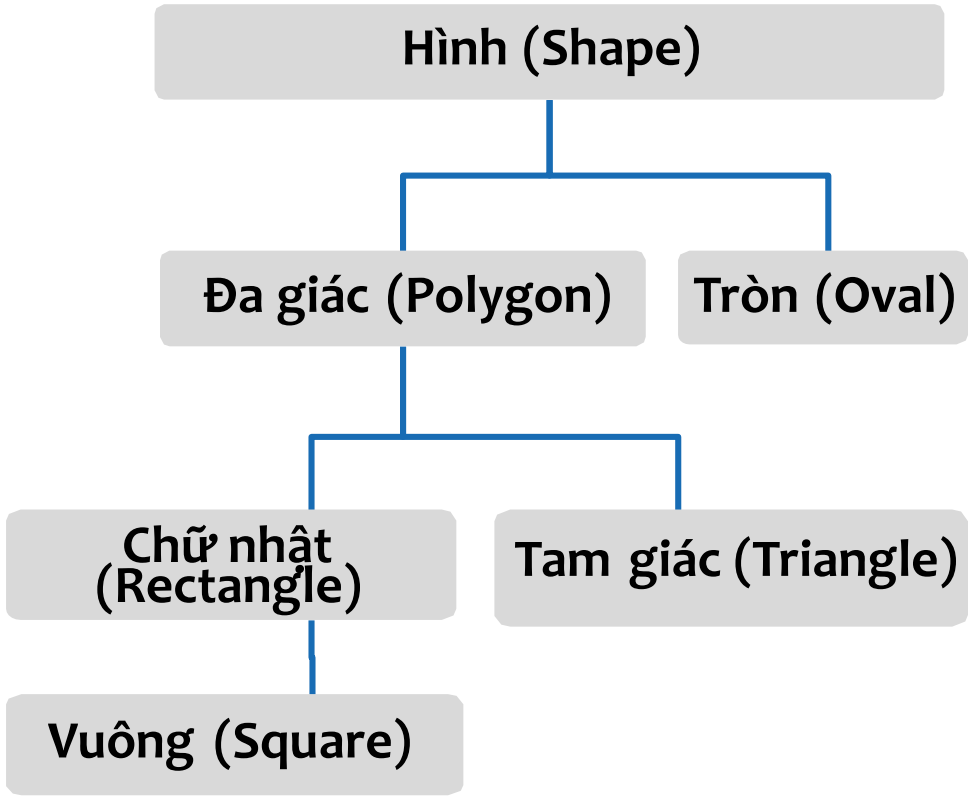
```
public class Person {

}

public class Programmer extends Person {

}

public class Dancer extends Person {

}

public class Singer extends Person {

}
```



Tính kế thừa (3)

Thể hiện kế thừa

```
public class Shape {…}

public class Polygon extends Shape {…}

public class Oval extends Shape {…}

public class Rectangle extends Polygon {…}

public class Triangle extends Polygon {…}

public class Square extends Rectangle {…}
```



Tính kế thừa (4)

Định nghĩa

- Kế thừa là cơ chế tạo **1 lớp mới** dựa trên kết quả từ **lớp đã có**
- Lớp con **ĐƯỢC PHÉP** sở hữu các tài sản (properties và methods) **không** phải là **private** của lớp cha

Được phép sử dụng
- **name, designation, learn()**

Không được phép sử dụng
- **display()**

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

  protected void learn() {}

  private void display() {}
}

public class Programmer extends Person {
   private String companyName;
}
```

Tính kế thừa (5)

Phương thức khởi tạo

- **Nếu lớp cha có nhiều constructor** thì lớp con được quyền chỉ định 1 constructor của lớp cha
- **Nếu lớp con không chỉ định** thì constructor mặc định của lớp cha sẽ tự động thực hiện
- **Constructor của lớp cha** được thực hiện trước rồi mới đến constructor của lớp con

Có lỗi vì không tìm thấy phương thức khởi tạo mặc định ở lớp Person

```
public class Person {
  protected String name;

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

}

public class Programmer extends Person {
  private String companyName;
}
```

Tính kế thừa (6)

Phương thức khởi tạo

```
public class Person {
  protected String name;
  protected String designation;

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

```
public class Programmer extends Person {
  private String companyName;

  public Programmer(String name) {
     super(name);
   }

}
```

- **super** được sử dụng để gọi phương thức khởi tạo của lớp cha
- super được sử dụng để truy cập các thuộc tính và phương thức của lớp cha (Khi cả **lớp cha** và **lớp con** có các **thành viên trùng tên**)

Tính kế thừa (7)

Phương thức ghi đè (Override Methods)

- Cơ chế ghi đè là việc khai báo các **phương thức trùng tên**, cùng danh sách **tham số** và **kiểu trả** về giữa **lớp cha** và **lớp con**
- Mục đích của việc ghi đè là để **sửa lại nội dung phương thức** của lớp cha tại lớp con

```
public class Person {
  public void display() {
    System.out.println("Name: " + name);
  }
}

public class Programmer extends Person {
  private String compName;
  @Override
  public void display() {
    super.display();
    System.out.println("Company:  " +
compName);
  }
}
```

Tính kế thừa (8)

Phương thức ghi đè (Override Methods)

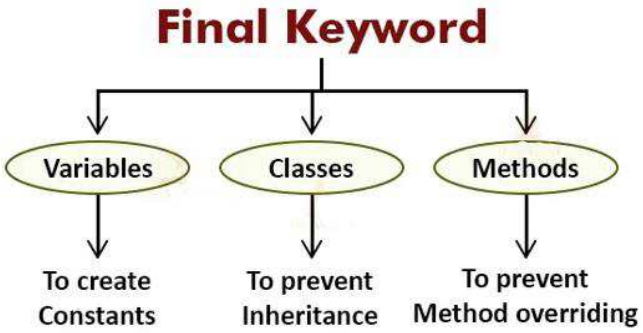
- Chỉ **được phép thay đổi phần thân** phương thức (implementation)
- **Không được phép thay đổi phần khai báo phương thức** (method name and signature)

Báo lỗi vì khác phần khai báo phương thức

```
public class Person {

  public void display() {
    System.out.println("Name: " + name);
  }
}

public class Programmer extends Person {
  private String compName;
  @Override
  public void display(String compName) {
    super.display();
    System.out.println("Company:  " + compName);
  }
}
```

3

Từ khóa final (Final Keyword)

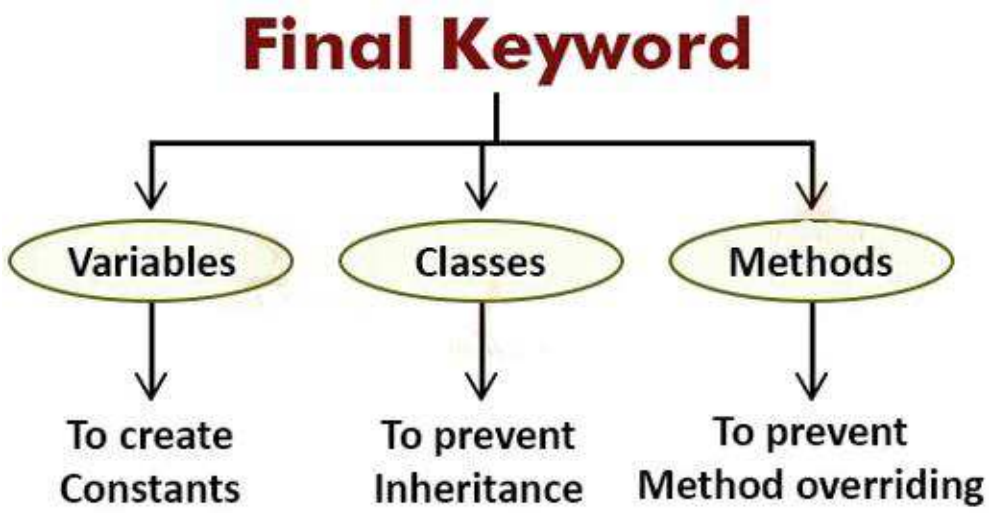


Từ khóa final (1)

Khai báo thuộc tính

- Một **thuộc tính** được khai báo với **final** thì chúng ta **không** thể thay đổi **giá trị** của nó một lần nữa và được gọi là **hằng số**

```
// Hằng số NUMBER
public final int NUMBER = 10;

// Hằng số RATE
public final float RATE = 2.5f;
```

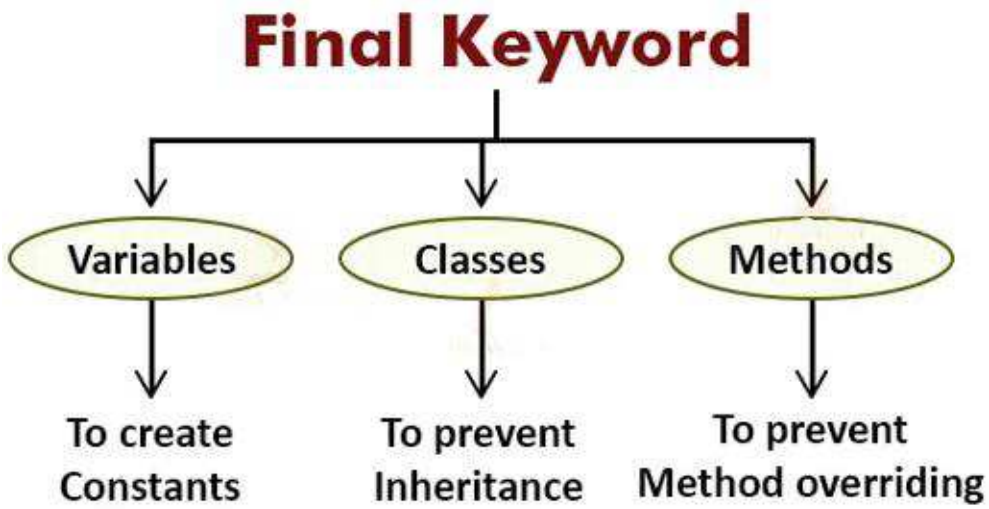


Từ khóa final (2)

Khai báo phương thức

- Một **phương thức** được khai báo với final thì lớp con **không thể ghi đè (override)** phương thức của lớp cha

```
public class MotorBike {
  final void run() {
    System.out.println("running");
  }
}

public class Honda extends MotorBike {
  void run() {
    System.out.println("Honda is running");
  }
}
```



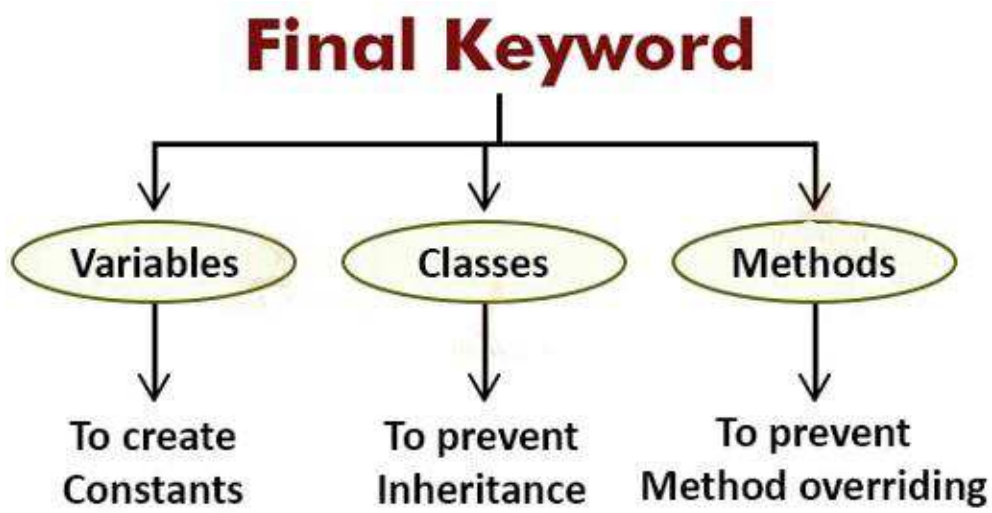
Báo lỗi vì phương thức run() không được phép ghi đè (override)

Từ khóa final (3)

Khai báo lớp

- Khi một **lớp (class)** được khai báo với từ khóa final thì nó **không** cho lớp khác **kế thừa**

```
public final class MotorBike {

 }

public class Honda extends MotorBike {
   void run() {
      System.out.println("Honda is running");
   }
}
```



Báo lỗi vì lớp MotorBike không được phép kế thừa

Resource Software Solution

4

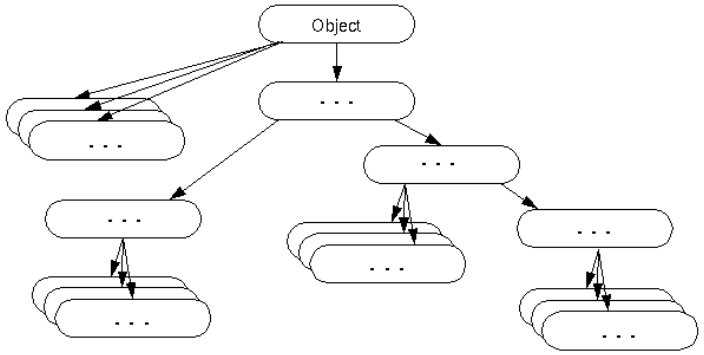
Lớp Object (Object Class)

Lớp Object (1)

Object class

- Mặc định lớp **Object** là lớp cha của tất cả các lớp
- Lớp Object được sử dụng nếu chúng ta muốn **tham chiếu** đến bất kỳ **đối tượng nào** có kiểu mà chúng ta **không biết**
- Ví dụ phương thức **getObject ()** trả về một đối tượng và nó có thể thuộc bất kỳ loại nào như **Nhân viên, Sinh viên,** v.v.

```
Object obj = getObject();
```

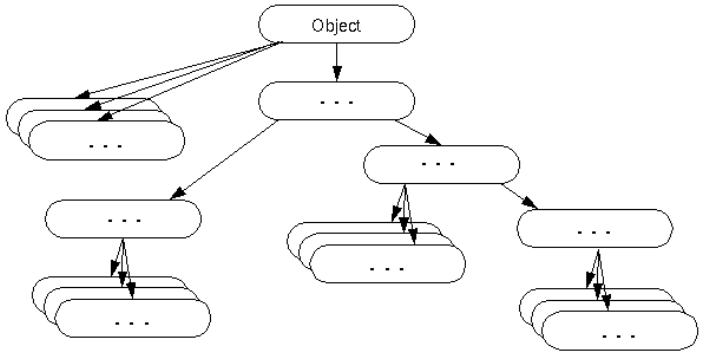


Lớp Object (2)

Object class

- Những phương thức của lớp Object

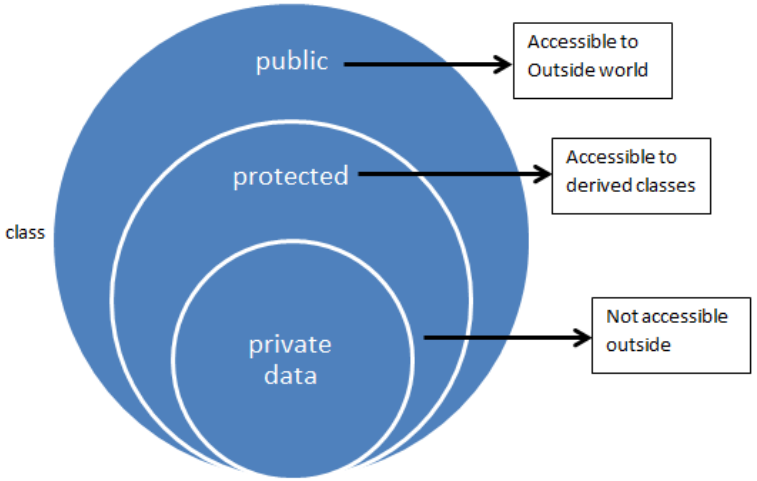
Phương thức	Miêu tả
public boolean equals(Object obj)	So sánh hai đối tượng
protected Object clone()	Tạo và trả về bản sao của đối tượng
public String toString()	Trả về chuỗi biểu diễn của đối tượng



Tổng kết

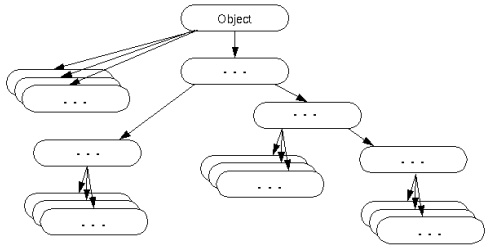
1. Tính đóng gói

- Bảo vệ dữ liệu (thuộc tính) bằng từ khóa private
- Cung cấp phương thức getter và setter



2. Tính kế thừa (đơn thừa kế)

- Lớp con kế thừa thuộc tính, phương thức không phải là private của lớp cha
- Khi kế thừa có thể override phương thức của lớp cha
- Sử dụng final để ngăn kế thừa



Thankyou!