Để làm được bài này bạn cần biết được 2 đặc điểm của tính kế thừa.

Lớp con không được thừa hưởng các thuộc tính và phương thức private từ lớp cha.

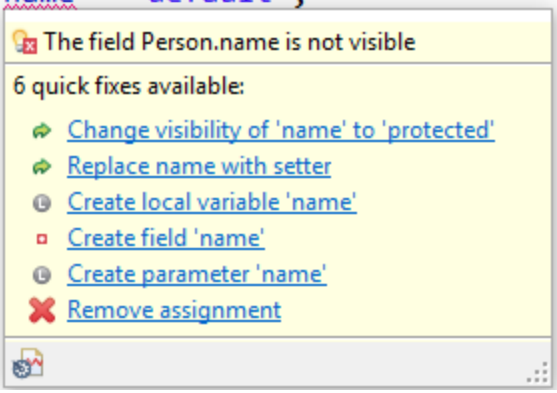
Trong kế thừa, các lớp con chỉ có thể thừa hưởng được các thuộc tính và phương thức có phạm vi truy cập `public, protected` và `default` (trong trường hợp lớp con và lớp cha cùng package). Nếu bạn truy xuất tới thuộc tính `private` của lớp cha từ lớp con thì chương trình sẽ báo lỗi giống như sau:

```
class Person {
    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}

class Student extends Person {
    public Student() {
        name = "default";
    }
}
```



Muốn truy xuất được tới các thuộc tính `private` của lớp cha thì bạn phải thông qua các setter và getter của lớp cha:

```java
class Person {
    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}

class Student extends Person {
    public Student() {
        setName("Default");
    }
}
```

Do đó trong bài này bạn không thể truy xuất trực tiếp tới thuộc tính `name` và `dob` của lớp `Person` từ lớp `Student`, thay vào đó bạn phải sử dụng các setter và getter.

Constructor của lớp con luôn gọi tới constructor của lớp cha

Constructor của lớp con luôn gọi tới constructor luôn gọi tới constructor của lớp cha, nếu bạn không chỉ rõ là cần gọi tới constructor nào của lớp cha thì lớp con sẽ luôn gọi tới constructor mặc định của lớp cha. Ví dụ:

```java
class Person{
	public Person() {
		System.out.println("Person constructor");
	}
}

class Student extends Person{
	public Student() {
		System.out.println("Student constructor");
	}
}

class Entry {
	public static void main(String[] args) {
```

```
                Student s = new Student();
        }
}
```

Kết quả khi chạy chương trình:

```
Person constructor
Student constructor
```

Có thể thấy constructor mặc định (constructor không tham số) của lớp cha đã được gọi cùng với constructor của lớp con. Do đó, khi lớp cha không có constructor mặc định mà lớp con không chỉ rõ cần gọi tới constructor nào của lớp cha thì chương trình sẽ báo lỗi giống như sau:

```
class Person {
    private String name;
    public Person(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}

class Student extends Person {
}
```

Implicit super constructor Person() is undefined for default constructor. Must define an explicit constructor

1 quick fix available:

Add constructor 'Student(String)'

Lúc này bạn phải dùng từ khóa `super` để chỉ cho lớp con biết cần phải gọi tới constructor nào của lớp cha giống chương trình sau:

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

```
        }

        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }
}

class Student extends Person {
        public Student(String name) {
                super(name);
        }
}

public class Entry {
        public static void main(String[] args) {
                Student s1 = new Student("Trung");
                System.out.println(s1.getName());
        }
}
```

Kết quả khi chạy chương trình:

```
Trung
```

Trong ví dụ trên `super(name)` chính là gọi tới constructor một tham số của lớp cha.

Từ khóa `super` khác từ khóa `this` ở chỗ từ khóa `super` sẽ tham chiếu tới lớp cha còn từ khóa `this` sẽ tham chiếu tới lớp hiện tại. Do đó bạn cũng có thể truy xuất tới các thuộc tính và phương thức của lớp cha bằng từ khóa `super`.