**Task 6 - Cơ bản về lập trình hướng đối tượng**

1. **Lớp và đối tượng**
2. **Khái niệm**

- **Đối tượng** là thực thể có thuộc tính và hành vi. Trong lập trình, thuộc tính được lưu trữ trong các trường và hành vi được lưu trong phương thức.

- **Lớp** là khuôn mẫu của một tập hợp các đối tượng có cùng thuộc tính và hành vi (phương thức). Lớp mô tả các đối tượng sẽ được tạo ra hay một kiểu để tạo ra các đối tượng.

1. **Tạo và sử dụng đối tượng**

- Khai báo lớp:

*<accessModifier> class <className> { //Khai báo thuộc tính, phương thức }*

- Mỗi lớp nên có một hàm khỏi tạo không có tham số:

*<accessModifier> <className>() { // Khởi tạo dữ liệu }*

- Khởi tạo đối tượng:

*<className> <objectName> = new <className>( //Tham số nếu có );*  
- Sử dụng đối tượng: *<objectName>.<attribute/method( //Tham số nếu có )>*

1. **Phương thức khởi tạo và nạp chồng**
2. **Phương thức khởi tạo (Constructor)**

- Constructor: phương thức đặc biệt dùng để khởi tạo đối tượng của một lớp. Nó có chức năng tạo ra đối tượng và khởi tạo các giá trị cần thiết cho đối tượng đó (thuộc tính,...).

- Constructor được gọi tự động khi một đối tượng được tạo bằng từ khóa **new**.

- Có 3 loại hàm khởi tạo:

* Hàm khởi tạo mặc định: không có tham số, được Java tạo tự động nếu không có bất kỳ hàm tạo nào được định nghĩ trong lớp.
* Hàm khởi tạo không có tham số: giống với hàm khởi tạo mặc định, tuy nhiên được lập trình viên định nghĩa và kiểm soát các trường dữ liệu được khởi tạo.
* Hàm khởi tạo có tham số: là hàm khởi tạo được định nghĩa để khởi tạo đối tượng với các giá trị được truyền vào thông qua tham số

1. **Nạp chồng (Overloading)**

- Nạp chồng là kỹ thuật cho phép nhiều phương thức có cùng tên nhưng khác nhau về tham số (số lượng tham số, kiểu dữ liệu,...). Ví dụ như constructor của các lớp trong Oop.

- Mục đích: tạo giao diện lập trình dễ hiểu và thuận tiện hơn cho người dùng, tận dụng tính đa hình của lập trình hướng đối tượng.

- Nạp chồng có nhiều lợi ích:

* Tăng khả năng đọc hiểu và rõ ràng của mã nguồn
* Tận dụng tính chất của kiểu dữ liệu
* Tận dụng tính đa hình
* Tăng khả năng mở rộng

1. **Thuộc tính và phương thức**
2. **Khai báo thuộc tính**

- Cú pháp:

*[<accessModifier>] <attributeType> attributeName [= <defaultValue>];*

- AccessModifier có thể là: public, private, protected. Nếu không được chỉ định, mặc định là private.

- Nếu không chỉ định defaultValue, Java sẽ tự gán giá trị mặc định theo kiểu dữ liệu đã khai báo thuộc tính.

1. **Khai báo và sử dụng phương thức**

- Cú pháp khai báo:

*<accessModifier> [static] <returnType> <methodName>([parameters]) {*

*//code*

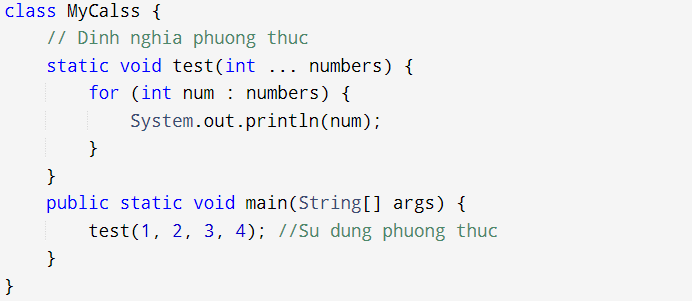
*}*

- AccessModifier có thể là: public, private, protected. Nếu không được chỉ định, mặc định là private.

- Tham số của các phương thức:

* Tham số truyền vào bên trong cặp ngoặc tròn (“()”), cách nhau bởi dấu phẩy
* Tham số tùy ý: số lượng tham số truyền vào là tùy ý. Các giá trị tham số được lưu trong mảng. Cú pháp: *dataType ... varName*

- Ví dụ về phương thức với tham số tùy ý:



- Gọi phương thức từ đối tượng: *<objName>.<method( [parameters] );*

1. **Access modifier**

|  |  |  |
| --- | --- | --- |
| **Access Modifier** | **Ý nghĩa** | **Mối quan hệ trong kế thừa** |
| public | Truy cập được ở mọi nơi trong chương trình | Phải là public trong tất cả các lớp con |
| private | Không được khai báo ở lớp cao nhất, chỉ có thể truy cập từ bên trong lớp mà nó khai báo | Lớp con không được kế thừa |
| protected | Có thể truy cập ở bên trong và bên ngoài package nhưng phải thông qua tính kế thừa | Phải là protected hoặc public trong tất cả lớp con |
| default | Khi không chỉ rõ loại access modifier nào thì Java đặt nó là default (package-private) và chỉ được truy cập bên trong package | Có thể private trong các lớp con |