-Kế thừa là một cơ chế để 1 class có thể bắt đầu từ 1 class khác(base class). Class mới có được thừa hưởng một số thuộc tính và phương thức từ class gốc. Kế thừa là 1 trong những nguyên tắc chính của lập trình hướng đối tượng. Nó cho phép lập trình viên xây dựng những hệ thống phân cấp class 1 cách tiện lợi và tái sử dụng lại code có sẵn.

-Để 1 class kế thừa 1 class khác ta sử dụng từ khóa "extends", cú pháp được sử dụng như sau:

class SuperClass { }

class SubClassA extends SuperClass { }

class SubClassB extends SuperClass { }

class SubClassC extends SubClassA { }

-Khi chúng ta định nghĩa 1 class con mới chúng ta cần truy cập vào các members hoặc constructors của class cha của nó. Java cung cấp từ khóa đặc biệt "super" để làm điều đó. Từ khóa này có thể được sử dụng trong một số trường hợp sau:

+truy cập vào các instance fields của class cha

+invoke methods của class cha

+invode constructors của class cha(có tham số hoặc không)

-Java provides an opportunity to declare a method in a subclass with the same name as a method in the superclass. This is known as method overriding. The benefit of overriding is that a subclass can give its own specific implementation of a superclass method.

-Method overloading is a programming concept that allows you to design numerous methods that share the same name but have distinct arguments. When a method is called, the language determines which version of the method to run based on the number, types, and order of the arguments given. With method overloading, you can declare different versions of a method with the same name, creating code that is more concise and easier to understand while still handling all sorts of inputs.

|  |  |  |
| --- | --- | --- |
| Method Overloading |  | Method Overriding |
| Overloading happens at compile time. |  | Overriding happens at **runtime** |
| Gives better performance because the binding is being done at compile time. |  | Gives less performance because the binding is being done at run time. |
| Private and final methods can be overloaded. |  | **Private** and **final** methods can not be overridden. |
| Return type of method does not matter in case of method overloading. |  | Return type of method must be the same in the case of overriding. |
| Arguments must be different in the case of overloading. |  | Arguments must be the same in the case of overriding. |
| It is being done in the same class. |  | Base and derived classes are required here. |
| Mostly used to increase the readability of the code. |  | Mostly used to provide the implementation of the method that is already provided by its base class. |

-Phương thức Main có thể overloading.

-boolean equals(Object obj): checks whether this object and another one are stored in the same memory address.

- int hashCode(): returns an integer hash code that is unique for each object (object's identity).

- Math restrictions:

+ **Reflexivity**

**+** **Symmetry**

**+** **Transitivity**

**+** **Consistency**

**+** **Non-nullity**

- Theo nguyên tắc mà *hashCode()* đưa ra thì 2 object bằng nhau thì phải có cùng mã *hashCode()*, vì vậy khi chúng ta override*equals()* method thì nhất định phải override *hashCode()* method.

-An abstract class is a class declared with the keyword abstract. It represents an abstract concept that is used as a base class for subclasses.

-Abstract classes have some special features:

+it's impossible to create an instance of an abstract class;

+an abstract class can contain abstract methods that must be implemented in non-abstract subclasses;

+it can contain fields and non-abstract methods (including static);

+an abstract class can extend another class, including an abstract one;

+it can contain a constructor.

-As you can see, an abstract class has two main differences from regular (concrete) classes: no instances and abstract methods.

-Abstract methods are declared by adding the keyword abstract. They have a declaration (modifiers, a return type, and a signature) but don't have an implementation. Each concrete (non-abstract) subclass must implement these methods.

-Note, static methods can't be abstract!

-In Java, an **interface** is a reference type that is similar to a class but has a number of distinctive features that make it different. An interface contains declarations of methods and serves as literally an interface between a class inheriting that interface and the other classes.

**-Abstract classes** and **interfaces** are both tools to achieve abstraction that allow us to declare abstract methods. We cannot create instances of abstract classes and interfaces directly, we can only do that through classes that inherit them.

-Since Java 8, an interface can have default and static methods that contain an implementation. It makes interface more similar to an abstract class. So, the important question is: what is the difference between interfaces and abstract classes?

-Below you can see a list of some important differences between these two concepts.

* an **abstract class** can have *abstract* and *non-abstract* instance methods while an **interface**can have *abstract* or *default* instance methods;
* an **abstract class** can extend another abstract or regular class and an **interface** can only extend another interface;
* an **abstract class** can extend only one class while an **interface** can extend any number of interfaces;
* an **abstract class** can have *final*, *non-final*, *static*, *non-static* *variables* (regular fields) while an interface can only have *static final variables*;
* an **abstract class** can provide an implementation of an interface but an **interface** cannot provide an implementation of an abstract class;
* an **abstract class** can have a constructor and an **interface** cannot;
* in an **abstract class**, the keyword abstract is mandatory to declare a method as an *abstract* one while in an **interface** this keyword is optional.

-Remember, a class **extends** another class, a class **implements** an interface, but an interface **extends** another interface.

-The provided list of differences is by no means complete. **Abstract classes** and **interfaces** have a lot of other differences but the main one is their purpose.

-Typically, interfaces are used to decouple the interface of a component (class) from the implementation while abstract classes are often used as base classes with common fields to be extended by subclasses. The picture below demonstrates the last statement.