**Java基础语法**

**Java修饰符**

访问控制修饰符

default, public, protected, private

非访问控制修饰符

final, abstract, static, synchronized

**Java关键字**

|  |  |  |
| --- | --- | --- |
| 类别 | 关键字 | 说明 |
| 访问控制修饰符 | private | 在同一类内可见，对变量和方法使用 |
| protected | 对同一包内的类和所有子类可见，对变量和方法使用 |
| public | 对所有类可见，对类、接口、变量和方法使用 |
| default（不写） | 在同一包内可见，对类、接口、变量和方法使用 |
| 类、方法和变量修饰符 | abstract | 声明抽象 |
| class | 类 |
| extends | 扩充、继承 |
| final | 最终值、不可改变的 |
| implements | 实现（接口） |
| interface | 接口 |
| native | 本地方法 |
| new | 新、创建 |
| static | 静态 |
| strictfp | 严格、精准 |
| synchronized | 线程、同步 |
| transient | 短暂 |
| volatile | 易失 |
| 程序控制语句 | break | 跳出循环 |
| case | 定义一个值以供switch选择 |
| continue | 继续下一次循环 |
| default | 默认 |
| do | 运行 |
| else | 否则 |
| for | for循环 |
| if | 如果 |
| instanceof | 实例 |
| return | 返回 |
| switch | 根据值进行选择 |
| while | while循环 |
| 错误处理 | assert | 断言表达式为真/假 |
| catch | 捕捉异常 |
| finally | 有没有异常都执行 |
| throw | 抛出异常 |
| throws | 声明一个异常可能被抛出 |
| try | 捕获异常 |
| 包 | import | 引入 |
| package | 包 |
| 基本类型 | boolean | 布尔型 |
| byte | 字节型 |
| double | 双精度浮点型 |
| float | 单精度浮点型 |
| int | 整型 |
| long | 长整型 |
| char | 字符型 |
| short | 短整型 |
| 变量引用 | super | 父类、超类 |
| this | 本类 |
| void | 无返回值 |
| 保留关键字 | goto | 是关键字，但不能使用 |
| const | 是关键字，但不能使用 |

**基本数据类型**

byte（8位有符号）

范围：-128～127（）

默认值：0

short（16位有符号）

范围：-32768～32767（）

默认值：0

int（32位有符号）

范围：-2147483648～2147483647（）

默认值：0

long（64位有符号）

范围：-9223372036854775808～9223372036854775807（）

结尾要加上L或l

默认值：0L

float（32位）

结尾加上F或f

默认值：0.0f

double（64位）

结尾加上D或d，可省略

默认值：0.0d

boolean

默认是false

char

**循环**

while

while (statement){

loop content;

}

do...while

do {

loop content;

}while (statement);

for

for (initial; statement; update){

loop content;

}

foreach

for (varable: saurce){

loop statement;

}

**条件语句**

if语句

if (statement){

content;

}

if...else语句

if (statement){

content1;

}

else{

content2;

}

else...if语句

if (statement1){

content1;

}

else if(statement2){

content2;

}

switch...case

switch (variable name){

case value1:

content1;

case value2:

content2;

default: //可选

content3;

}

**Number**

public boolean euqals(Object object);

用于比较两个值是否相同，而”==”是用来比较两个地址是否一致

String toString();

返回一个字符串表示的Number

**Math**

Math.abs(任何原生类型);

返回绝对值

Math.round(Number);

返回四舍五入的int/long类型的值

Math.min(p1, p2); / Math.max(p1, p2);

Math.random();

不接受任何参数，返回0-1之间的一个随机数

**Character**

Character.isLetter(char ch);

判断指定字符是否为字母，返回boolean值

Character.isDigit(char ch);

判断指定字符是否为数字，返回boolean值

Character.isWhitespace(char ch);

判断是否为空白字符（空格，tab，换行），返回boolean值

Character.isUpperCase(char ch); / Character.isLowerCase(char ch);

判断是否为大/小写字母，返回boolean值

Character.toUpperCase(char ch); / Character.toLowerCase(char ch);

大小写转换

Character.toString(char ch);

把指定字符转为一个长度为1的字符串

**String**

Immutable, constant pool, heap

String.charAt(int n);

返回位置n处的字符

String.replace(char old, char new);

把字符串中的所有old替换成new

String.toCahrArray();

把这个字符串转换为一个字符Array

String.contains(charSequence chars);

判断是否包含指定的字符序列

String.isEmpty();

判断该字符串是否为空

**String Builder & String Buffer**

Mutable

StringBuilder is faster than StringBuffer

StringBuilder is not Thread Safety, it is non-synchronized

StringBuffer is Thread Safety, it is synchronized

**final**

final means it cannot be changed. If final be used to decorate a class, this class cannot be inherited; if final be used to decorate a method, this method cannot be override; if final be used before a variable, so this variable cannot be modified.

finally be used in try...catch block, and finally part will always be executed eventually

finalize be used in garbage collection

**OOP**

Inheritance

Inheritance in Java is a concept that acquires the properties from one class to other classes; for example, the relationship between a superclass and subclass.

格式

class Superclass {}

class Subclass extends Superclass {}

用super()调用父类里的方法，用this()调用自身类里的方法

子类不能继承父类的constructor，如果父类的constructor有参数的话，子类里需要用super()来传参数给父类constructor

override & overload

override

When a method signature (name and parameters) is the same in the superclass and the child class, it's called Overriding, a final method cannot be override.

overload

When two or more methods in the same class have the same name but different parameters, it's called Overloading.

polymorphism

Polymorphism in Java is the ability of an object to take many forms. To simply put, polymorphism in java allows us to perform the same action in many different ways.

3 ways to implement polymorphism in Java

override

interface

abstract class & abstract method

abstract class

abstract class is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class)

abstract method: can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from)

use an abstract modifier to define an abstract class/method

encapsulation

encapsulation refers to the bundling of data with the methods that operate on that data. Encapsulation is used to hide the values or state of a structured data object inside a class, preventing unauthorized parties' direct access to them

interface

An interface in Java is a blueprint of a class. It has static constants and abstract methods. The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

enum

An enum is a special "class" that represents a group of constants (unchangeable variables, like final variables)

enum.values()

返回一个包含enum里面所有值的array

enum.value.ordinal()

可以找到每个枚举常量的索引，就像数组索引一样

enum.value.valueOf()

方法返回指定字符串值的枚举常量

**Abstract class**

Abstract class is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class). Abstract method can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

**Abstract class vs Interface**

Type of methods: Interface can have only abstract methods. An abstract class can have abstract and non-abstract methods.

Final Variables: Variables declared in a Java interface are by default final. An abstract class may contain non-final variables.

Type of variables: Abstract class can have final, non-final, static and non-static variables. The interface has only static and final variables.

Implementation: Abstract class can provide the implementation of the interface. Interface can’t provide the implementation of an abstract class.

Inheritance vs Abstraction: A Java interface can be implemented using the keyword “implements” and an abstract class can be extended using the keyword “extends”.

Multiple implementations: An interface can extend another Java interface only, an abstract class can extend another Java class and implement multiple Java interfaces.

Accessibility of Data Members: Members of a Java interface are public by default. A Java abstract class can have class members like private, protected, etc.

**design pattern**

singleton

singleton make sure one single class which create its own object and make sure there is only one class be created. and this class also provide a way to access the object directly without instantized the object of this class.

pros: There is only one instance in memory, which reduces memory overhead, especially when instances are created and destroyed frequently. Avoid multiple occupation of resources

cons: it carries state of object, and it prevents dependency injection and cannot be unit tested

steps:

1. private Constructor
2. it carries state of object
3. it carries state of object
4. it carries state of object

factory

In the factory pattern, we do not expose the creation logic to the client when creating the object, and we use a common interface to point to the newly created object.

pros: 1. A caller wants to create an object, as long as it knows its name. 2. High scalability. If you want to add a product, you only need to extend a factory class. 3. The specific implementation of the shielding product is shielded, and the caller only cares about the interface of the product.

cons: Each time a product is added, a concrete class and object implementation factory need to be added, which doubles the number of classes in the system, increases the complexity of the system to a certain extent, and also increases the dependency of the system's concrete classes. This is not a good thing.

decorator

Decorator Pattern allows adding new functionality to an existing object without changing its structure

pros: The decorator class and the decorated class can develop independently and will not be coupled with each other. The decorator pattern is an alternative pattern to inheritance. The decorator pattern can dynamically extend the function of an implementation class.

cons: multi-layer decoration is complicated

https://www.runoob.com/design-pattern/decorator-pattern.html

**Iterator vs Enumeration**

Iterator is modifiable but Enumeration is un-modifiable

**Exception Handling**

**Overload vs Override**

When two or more methods in the same class have the same name but different parameters, it's called Overloading. When the method signature (name and parameters) are the same in the superclass and the child class, it's called Overriding.

**Static**

a static variable is a variable that has been allocated "statically", meaning that its lifetime (or "extent") is the entire run of the program. This is in contrast to shorter-lived automatic variables, whose storage is stack allocated and deallocated on the call stack; and in contrast to objects, whose storage is dynamically allocated and deallocated in heap memory. When you want to have a variable that always has the same value for every object of the class, forever and ever, make it static. If you have a method that does not use any instance variables or instance methods, you should probably make it static.

A static method is a method that belongs to a class rather than an instance of a class. The method is accessible to every instance of a class, but methods defined in an instance are only able to be accessed by that object of a class. A static method is not part of the objects it creates but is part of a class definition. Unlike instance methods, a static method is referenced by the class name and can be invoked without creating an object of class. In simpler terms, they are methods that exist even if no object has been constructed yet and that do not require an invocation object. A static method can only access static data members and static methods of another class or same class but cannot access non-static methods and variables. Also, a static method can rewrite the values of any static data member.

A static class is basically the same as a non-static class, but there is one difference: a static class cannot be instantiated. In other words, you cannot use the new operator to create a variable of the class type.

**HashMap**

there are lots of buckets/bins in map, when we call put() method, java will hash the key and generate a hash code, calculate the index, and get the index will determine which bucket will hold this key-value pair. the java will call equals() method to check all keys in bucket, if matched, the exist key-value will be replaced, otherwise, the new key-value will be inserted in.

when we call get() method, Firstly, it finds the hashcode of the key to locate the bucket, Secondly, it scans the nodes, calling the equals method to find if the node’s key equals the current key. If end of the list is reached, the key is decided to not exist, and null will be returned.

**HashMap vs Hashtable**

HashMap is not synchronized, which means HashMap is not thread safe, HashMap is faster than Hashtable and HashMap accept null value as a key.

Hashtable is synchronized and is thread safe, so it slower than HashMap.