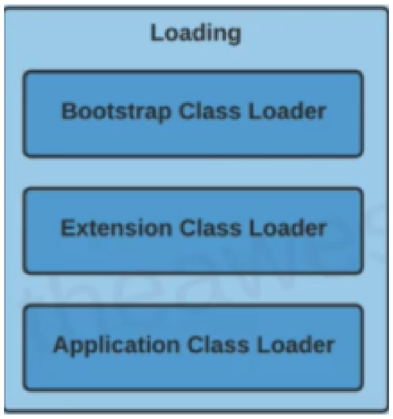
Java Basic-Jan 27

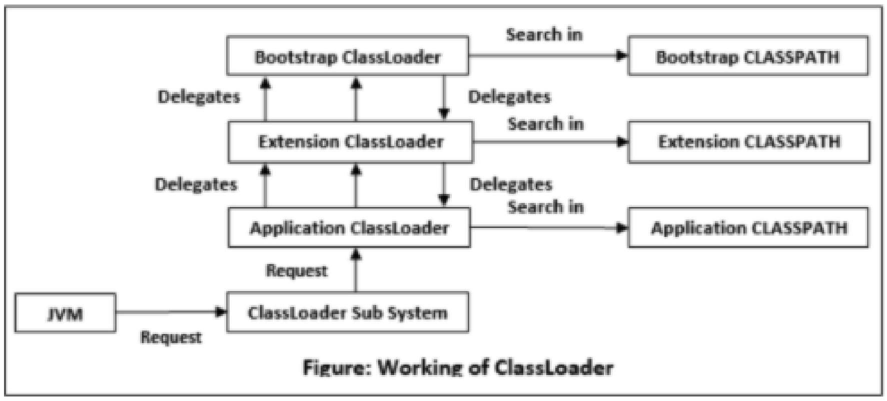
10. Class Loader



BootStrap Class Loader - Java package(java.lang)

Extension Class Loader - extension classes (dependence, like JDBC, ODBC)

Application Class Loader - classes in application classpath (like classes written by ourselves)



Process to load all the classes:

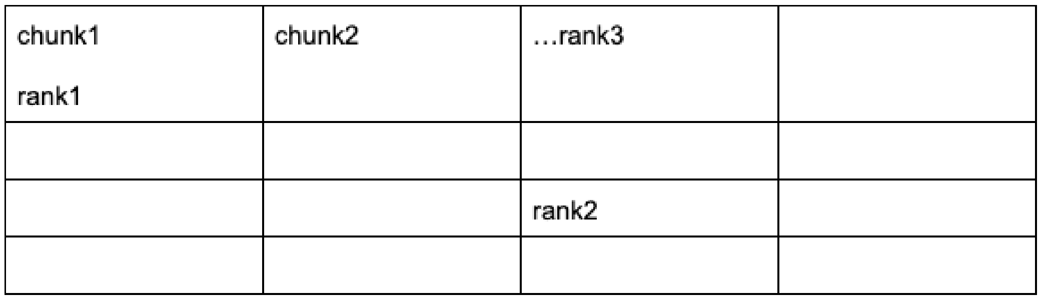
1. JVM → request → Classloader Sub System → (delegate)Application →(delegate) Extension → (delegate)Bootstrap

   super class relationship

      Bootstrap → Extension → Application

11. Garbage Collector

- serial GC (single thread)
  - entire application will stop, when serial GC do the garbage collection
  - use the command to operate it
  - XX:+UseSerialGC
- parallel GC (default)
  - multiple thread do the garbage collection
  - XX:UseParallelGC
- G1 GC
  - divide heap into equal sized chunks (prioritize which one will do the process first)
  - XX:++G1GC



chunk1 rank1	chunk2	…rank3	
		rank2	

do the parallel GC at chunk1 first, and chunk2, and then chunk3 by prioritized order

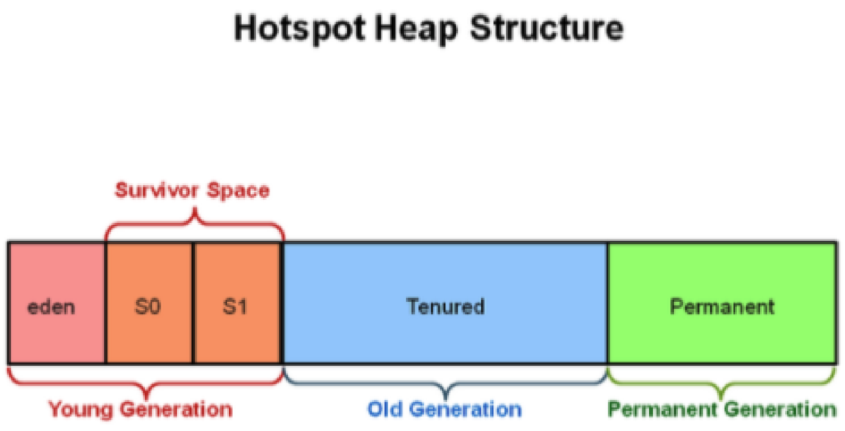
CMS(concurrent Mark Sweep)

- deprecated since Java 9
- completed removed in Java 14

GC process - **the heap can be divided to three spaces**

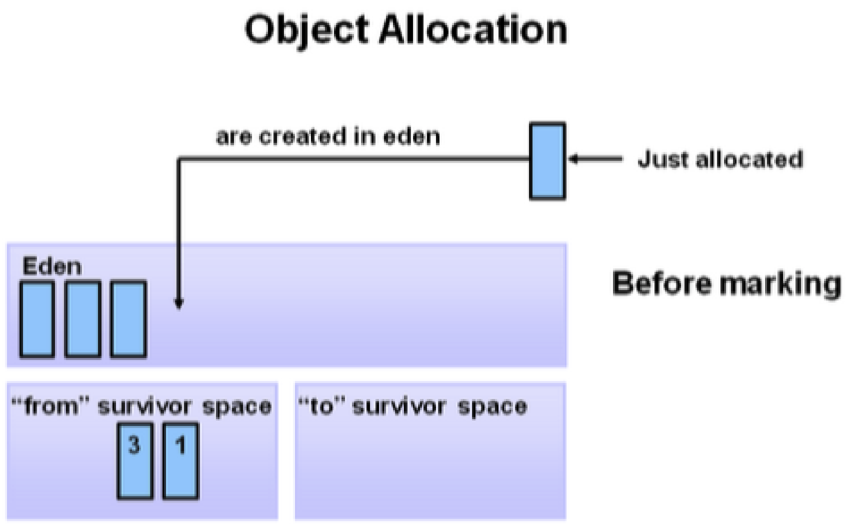
Collecting garbage from Young space (consisting of Eden and Survivor spaces) is called a Minor GC.

Major GC is **cleaning the Old space**. Full GC is cleaning the entire Heap – both Young and Old spaces.



Young Generation

- eden
- S0 (Survivor 0)
- S1 (Survivor 1)

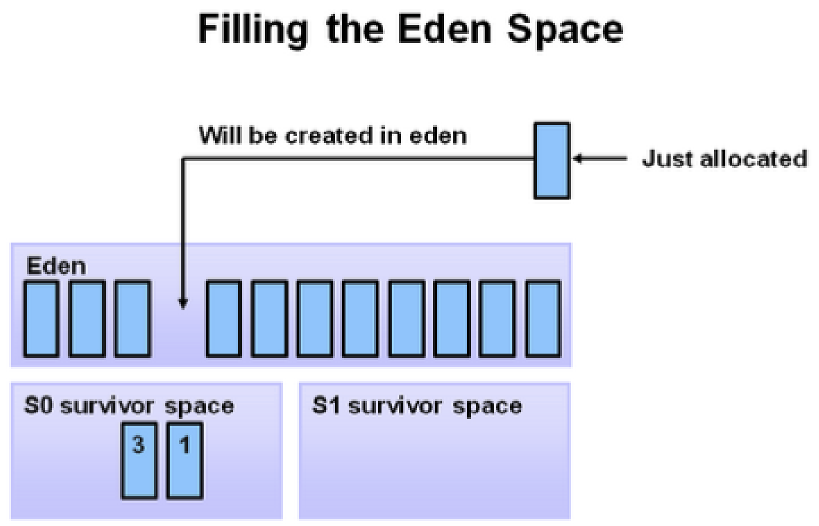


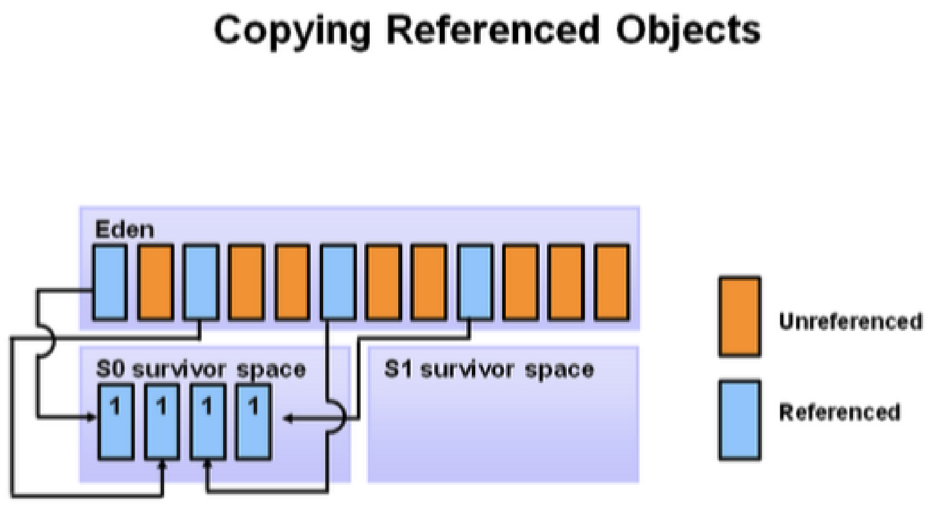
a. any new objects will be allocated(分配) to Eden first

b. if Eden area will be filled, it will trigger GC,

    i. objects are not referenced, they will be removed.

ii. objects are referenced, they will be remove to survivor space

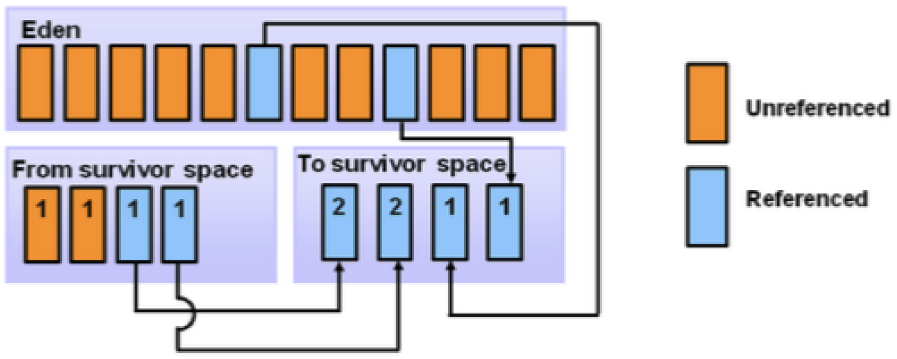




Unreferenced objects(orange) will be removed.

aging (使变老)

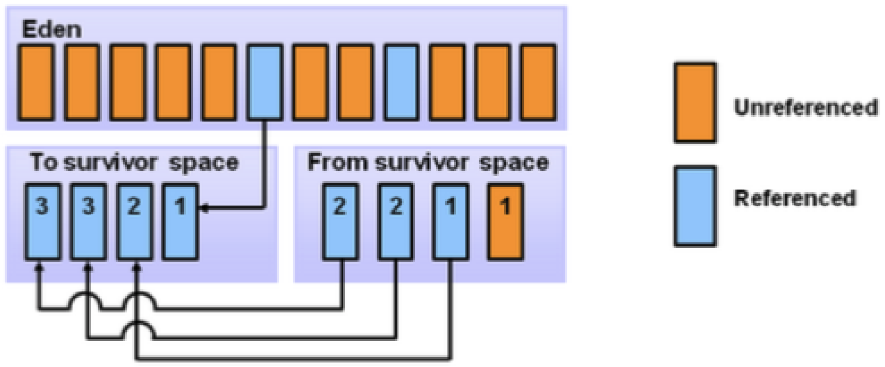
Object Aging



WhenEden and S0 are filled will trigger GC again.

All referenced objects will move to S1 space.

Additional Aging

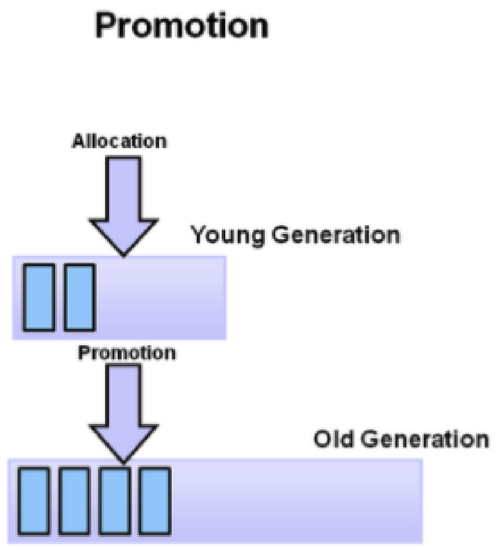


When Eden and S1 are filled will trigger GC again.

All referenced objects will move to S0.

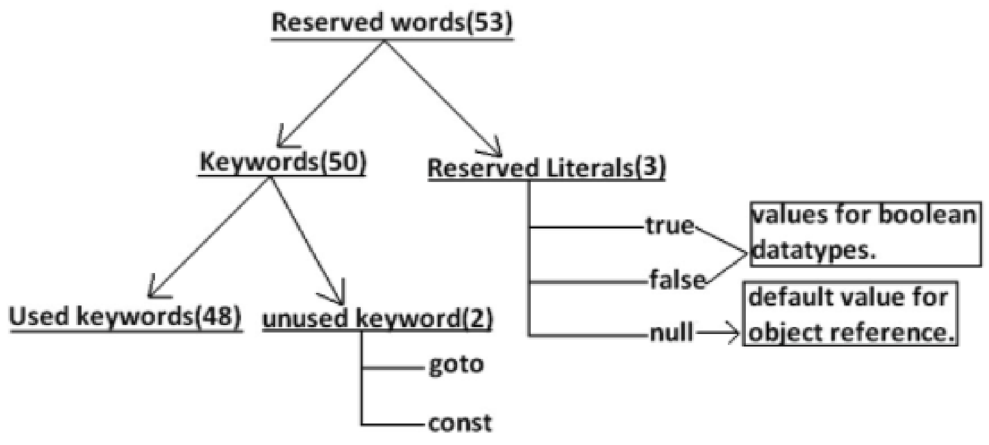
We can set a threshold (age), like 8.

When age is larger than 8, the object will move to old generation(tenured-终身职位) - middle part of first picture



Java do auto-GC

12. Keywords



/
———————————————————————————————————/

- Reserved words for **data types**:
    - byte
        - declare a variable as a numeric type. This ranges from -128 to 127.
    - short
        - to declare variables, contains minimum value of -32,768 and a maximum value of 32,767
    - int
        - declare a variable that can hold a 32-bit signed two's complement integer.
    - long
        - used to store 64-bit integer value
    - float
        - used to declare a variable which can store a floating point value.
    - double
        - declare a floating point variable as a numeric type, hold a 64-bit floating point number.
    - char
        - a data type that is used to store a single character
    - boolean
        - declare a floating point variable as a numeric type

- Reserved words for **flow control**:
    - if
        - a block of Java code to be executed if a condition is true
    - else
        - block of Java code to be executed if a condition is false
    - switch
        - a multiple-branch statement that executes one statement from multiple conditions.
    - case

        - a conditional label which is used with the switch statement.
        - e.g.

```
switch(number) {
  case 1 :
  System.out.println("1");
  break;
  case 2 :
  System.out.println("2");
  break;
}
```

    - default
        - an access modifier, If didn't assign any access modifier to variables/methods/constructors/classes by default, it is considered as default access modifier.
        - accessible within the package only.
    - for
        - used for loop, it executes a block of statements until the boolean expression returns true.
    - do
        - used in control statement to declare a loop. It provides a repetitive task as long as the condition specified with the while keyword is true; then it exits from the loop when the specified condition is founded as false.
    - while
        - used to iterate a part of the program repeatedly until the specified Boolean condition is true.
    - break
        - used to terminate loops and switch statements
    - continue
        - used to skip the current iteration of a loop
    - return
        - used to exit from a method, with or without a value.

- Keywords for **modifiers**:

- public
    - declares a member's access as public.
    - visible to all other classes
- private
    - declares a member's access as private.
    - only visible within the class
- protected
    - declared as protected can be accessed from: Within the same class. Subclasses of same packages.
- ~~static~~
- ~~final~~
- abstract
    - modify class: An abstract class is a restricted class that cannot be used to create objects
    - modify method: An abstract method can only be used in an abstract class, and it does not have a body.
- synchronized
    - The process of allowing only a single thread to access the shared data or resource at a particular point of time
    - No other thread can enter into that synchronized block until the thread inside that block completes its execution and exits the block.
- native
    - applied to a method to indicate that the method is implemented in native code using JNI
    - applicable only for methods, and we can’t apply it anywhere else. The methods which are implemented in C, C++ are called native methods or foreign methods.
    - Native modifier indicates that a method is implemented in platform-dependent code, often in C.
- strictfp
    - used in java for restricting floating-point calculations and ensuring the same result on every platform while performing operations in the floating-point variable.

- transient
  - used to avoid serialization
  - f any object of a data structure is defined as a transient , then it will not be serialized. Serialization is the process of converting an object into a byte stream.
- volatile
  - used to modify the value of a variable by different threads.
  - used to make classes thread safe.
  - multiple threads can use a method and instance of the classes at the same time without any problem.

- Keywords for **exception handling**:
  - ~~try~~
  - ~~catch~~
  - ~~finally~~
  - ~~throw~~
  - ~~throws~~
  - assert
    - used to declare an expected boolean condition in a program.
    - If the program is running with assertions enabled, then the condition is checked at runtime. If the condition is false, the Java runtime system throws an AssertionError.

- **Class** related keywords:
  - class
    - a collection of related variables and/or methods.
  - package
    - declares a 'name space' for the Java class.
    - mechanism to encapsulate a group of classes, sub *packages* and interfaces.
  - import
    - used to make the classes, interfaces, and other members of another package accessible to the current package.

    - extends
        - the child class inherits or acquires all the properties of the parent class.
    - implements
        - used to implement an interface.
    - interface
        - an abstract type that is used to specify a behavior that classes must implement.

- Object related keywords:
    - new
        - a Java operator that creates the object.
        - The new operator is followed by a call to a constructor.
    - instanceof
        - a binary operator used to test if an object (instance) is a subtype of a given Type.
        - It returns either true or false.
    - ~~super~~
    - ~~this~~

/
——————————————————————————————————————-/

Final

- final variable
    - create constant variable

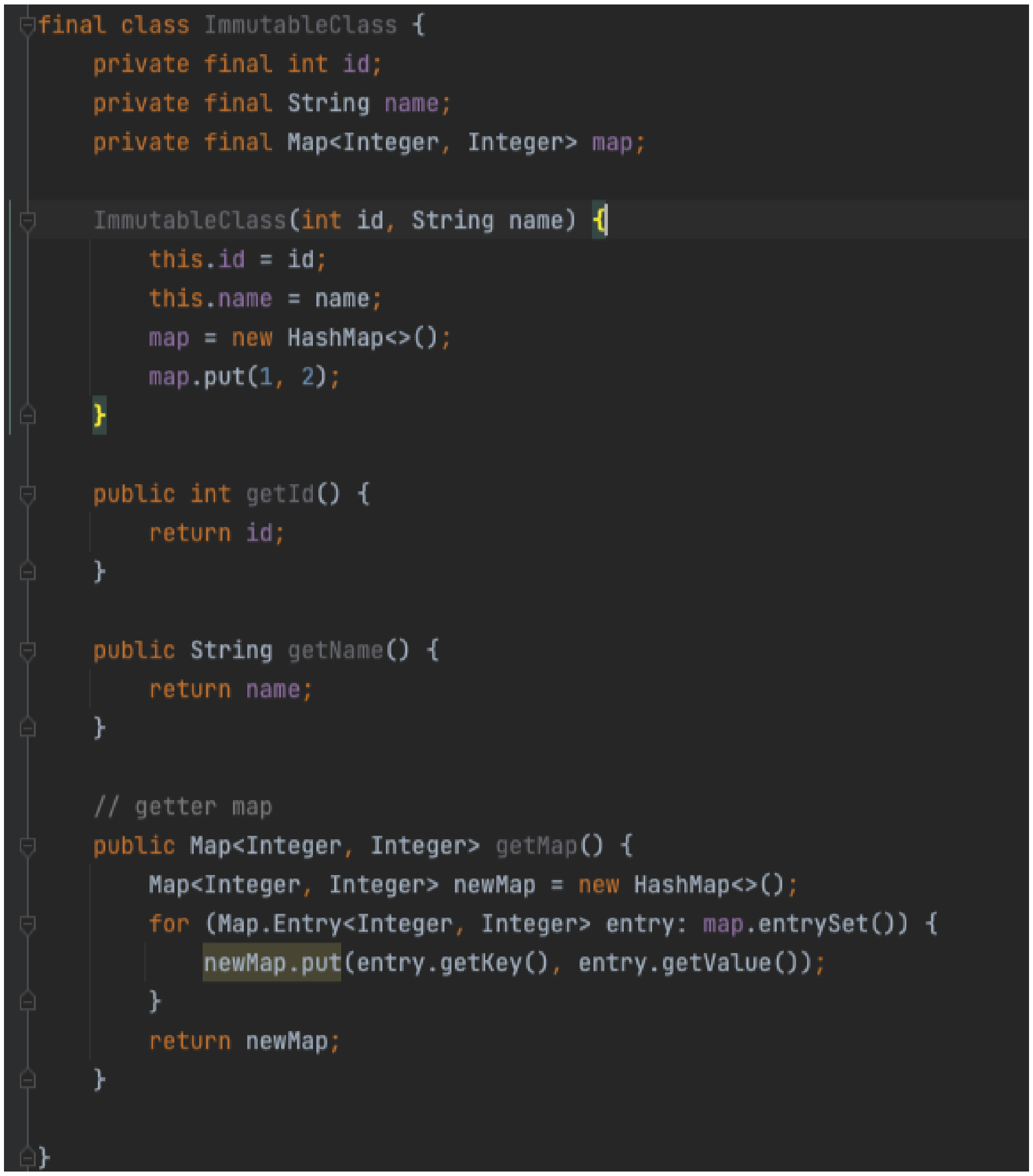
      when the variable is created, it can not be changed.
    - must be initialized (can be initialized in constructor)
- final method
    - represent the method can not be overridden
- final class
    - can’t be inherited

```
// final variable
final int a = 2;
a = 3; // compile error
//-------------------

final int b;
System.out.println(b); // not compile error, but when run,
                      // throw runtime error: b has not been initailized
//--------------------

final List<Integer> list = new ArrayList<>();
list.add(1);
System.out.println(list.size()); // it's work, we can change the internal
                                // elements, even though the List is final
list = new ArrayList<>();       // will be forbiden(not be allowed).

//--------------------
// final method

class A {
  final void method() {
  }
}

class B extend A {
  @Overrride
  void method() { //compile error
  }
}
//-------------------
//final class
final class A {}
class B extend A {} // compile error
```

immutable class

how to create/customize an immutable class

- make the class to be final
- all the variables to be final and private
- no setter
- return deep copy of the collections for getter
  - because for example a field like Map, when return it without deep copy, it can be modify(put, delete) - - return a reference



```java
final class ImmutableClass {
    private final int id;
    private final String name;
    private final Map<Integer, Integer> map;

    ImmutableClass(int id, String name) {
        this.id = id;
        this.name = name;
        map = new HashMap<>();
        map.put(1, 2);
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    // getter map
    public Map<Integer, Integer> getMap() {
        Map<Integer, Integer> newMap = new HashMap<>();
        for (Map.Entry<Integer, Integer> entry: map.entrySet()) {
            newMap.put(entry.getKey(), entry.getValue());
        }
        return newMap;
    }

}
```

static

use for memory management

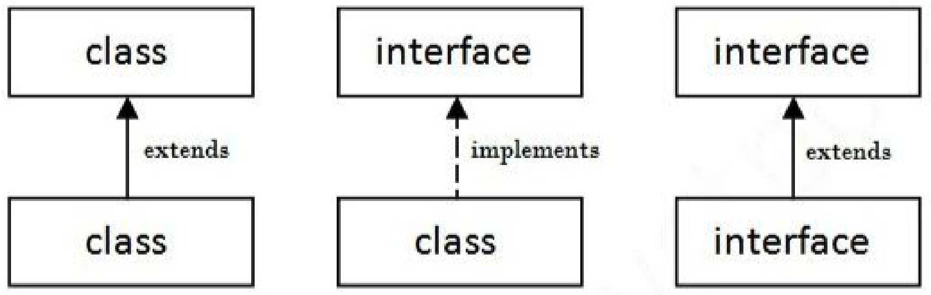
all the static info(variable, method...) are stored on the method area(JVM model)

modify (修饰) block/variable/class/method

- blocks
- variable
    - object belongs to the class

- methods
    - method belongs to the class,
    - do not need to create object to invoke method, just invoke the method directly
    -
- classes

implements vs. extends



this() vs. super()

final vs. finally vs. finalize()

finalize() used in garbage collection

Homework 3

keywords

13. OOP

4 aspects in Java

- **Abstraction** - process to hide the internal implementation
    - just expose the functionality to user, don't need to expose the internal application to user
    - achieve the abstraction

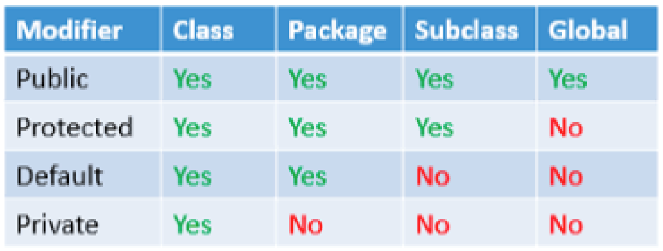
        - abstract class
        - interface

```
abstract class Shap { // abstract class can leave/implement abstract method
  int area;
  abstract public int getArea(); // if any class extends the Shap class
                                // it should override the abstract method
                                // show what is it the class
  public void method() {
    System.out.println("method imple,emtation");
  }
}

interface ShapeFunction { // shows the ability it has
                          // what funcationality it can provide
  void draw();
}
```

Q: What is abstract class? What is interface? What’s the difference?

- **Encapsulation - process to wrapping the code to a single unit**
    - declare the variable as private
    - declare setter and getter
- **Inheritance**
    - extends (only one class, single inheritance)
    - implements (multiple interface)
- **Polymorphism**
    - override
    - overloading
        - method names are same, but the arguments are different
        - return type can be anything, it doesn’t matter

Access Modifier

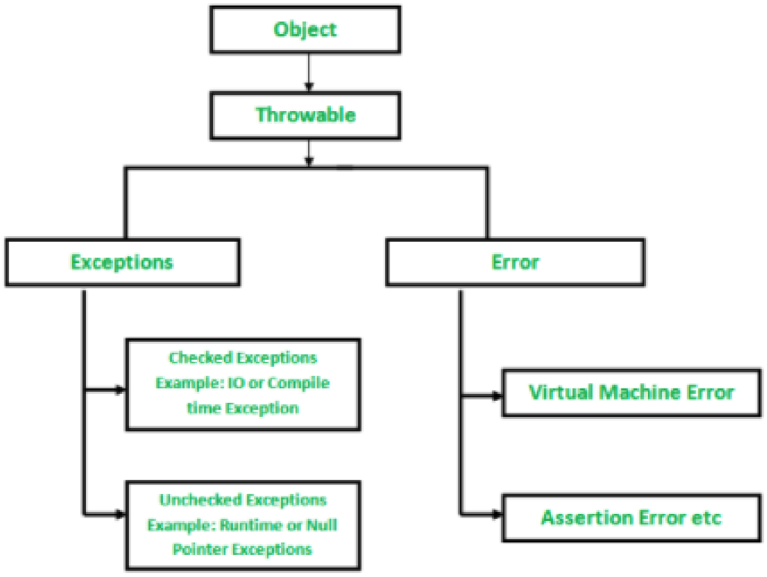


Modifier	Class	Package	Subclass	Global
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	No
Default	Yes	Yes	No	No
Private	Yes	No	No	No

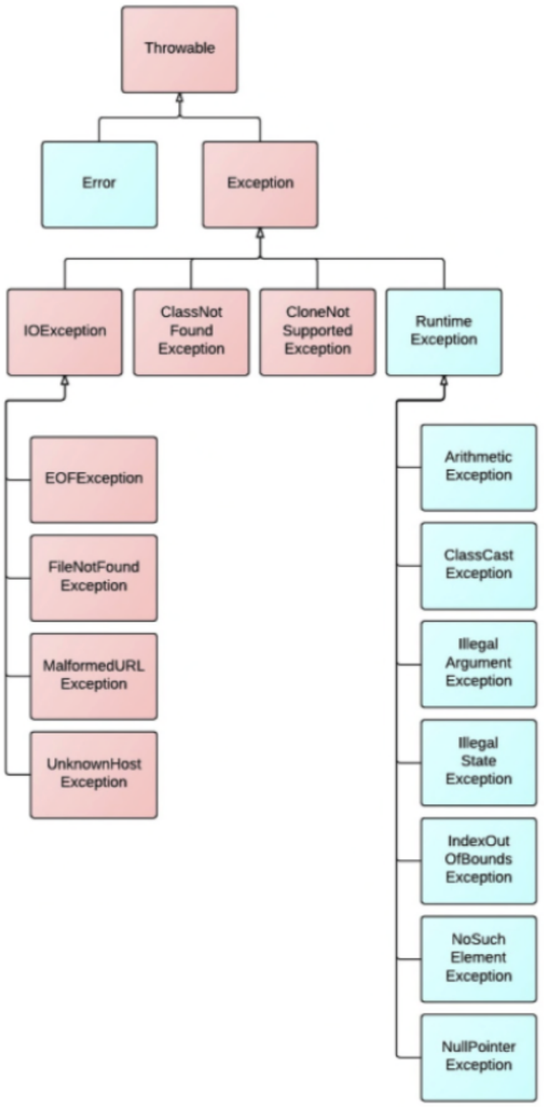
define the scope

globally accessible

14. Exception


Object
Throwable
Exceptions
Error
Checked Exceptions Example: IO or Compile time Exception
Unchecked Exceptions Example: Runtime or Null Pointer Exceptions
Virtual Machine Error
Assertion Error etc

all exceptions and error extends Throwable



Exceptions

1. Unchecked Exceptions (Runtime Exceptions)
    - ArrayStoreException, NullPointException, ArrayIndexOutOfBoundException...
    - we don't have to handle it
2. Checked Exceptions (compile time)
    - classNotFountException, IOException, SQLException….
    - handle
        - try catch (block)
        - throws (keyword)

        - throws the exception to the caller function (the caller function will handle the exception)

```
public void handleException() {
  try{
    int a = 0;
  } catch (SQLException) {
      rollback(); // in DB, do some rollback, if have errors
  } finally {
    connect.close(); // use finally to do some close operation
                     // close the connection
  }
}
//Tip: if try block has not a SQL relative codes, we cannot catch
// SQLException directly, it will show compile error

//---------------------------------------------------------
public void handleException2() throws Exception {
  // it will throw the Exception to its caller who invokes it
}

//for example
{
  //  hanleException2(); // directly invoke the function will
                          // trigger compile error
  // two solutions 1. try catch 2. thorws it in the signature of block
  // 套娃？！
  try {
    handleException2();
  } catch (Exception ex) {
    // do something
  }
}
```

Q: what’s the difference between throw and throws?

- throw is a keyword to be used to throw some new exceptions
- throws is a keyword to be used in function’s signature, throw the exception to the caller function

normally we throw the unchecked exceptions ?

Q: can we throw an error?

~~point: error needs to be solved before compiling~~

- yes, we can throw new Error();
- throw keyword can throw any class extends throwable, even we can throw any class who is customized by ourselves.

Customize Exception

```
class MyException extends Exception {
  public MyException() {
    super(); // load all the super functions
  }
  public MyException(String msg) {
    System.out.println(msg);
  }
}
```

Multiple catch (like if else statement)

- if try block has multiple exceptions need to be handled
- use multiple catch to handle them
  - scope/order from small to large (e.g IOException → Exception)
  - the former one should not be the **super class** of the latter

  ```
  try {
  // business logic
  } catch (IOException ex) {

  } catch (Exception ex) {

  } catch () {}
  catch () {}
  catch () {}
  catch () {}
  catch () {}
  // if there is ten or more exceptions need to be handled
  // don't need to write catch ten times

  // improvement in Java 7
  try {
    Connection con = DataDriver.getConnect();
    // some logics
  // some logics
  // some logics // something error in this logic
  // some logics

  } catch (IOException | SQLException ex ...) { // "|" pipe
    // exception handling
  ```

```
  // rollback
} finally {
if (con != null) con.close(); // free up the resource occupied by connection
}
```

try with resources

```
// use Parenthesis
// put the connection into Paraenthesis （bracket 括号）
try (
// connection area
Connection con = DataDriver.getConnect();
Statement stm = con.createStatement();
PreparedStatement ps = con.createPrepareStatement();
) {
// some logics
// some logics
// some logics
// some logics
} catch (ex ) {

}
```

don’t need to write finally block, cuz all codes in try parenthesis will be implement AntoClosable Interface.

in the interface, will be override close() method