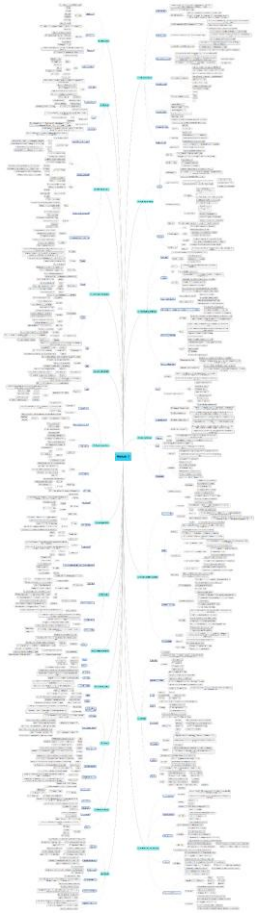
Module 2

Module 2

Overview



1. Tổng quan Java

- Khái niệm Java
  - Là ngôn ngữ lập trình hướng đối tượng , bậc cao có thể chạy trên mọi nền tảng
  - Sở hữu bởi Oracle
  - Đặc điểm của Java
    - đa nền tảng (Write Once, Run Anywhere: viết một lần, chạy bất cứ đâu
    - sử dụng rộng dãi
    - Bảo mật cao
- JDK, JRE, JVM
  - Java Development Kit (JDK): bộ công cụ phát triển java :

  - Bao gồm :

  + Trình biên dịch (compiler)=> javac
  + Debuger
  + JRE: môi trường thực thi java

  - Java Runtime Environment (JRE): môi trường thực thi java
    - Chứa các thư viện
    - JVM : máy ảo java
  - Java Virtual Machine (JVM): máy ảo java
    - Chuyển đổi java bytecode thành ngôn ngữ máy
    - Mỗi một hệ điều hành sẽ có một máy ảo riêng

1. Tổng quan Java(1)

- Biên dịch và thông dịch
  - Được sử dụng để chuyển đổi mã nguồn thành mã máy có thể thực thi được
  - Biên dịch (Compilation):là quá trình chuyển đổi mã nguồn java (.java) thành mã bytecode (.class) bằng trình biên dịch javac
    - Javac: Java Compiler
  - Thông dịch (Interpreter): là quá trình chạy mã bytecode (.java) bằng máy ảo (JVM)
    - JIT: Just In Time
- Các kiểu dữ liệu
  - Kiểu nguyên thủy (Primitive)
    - byte - Số nguyên, 1 byte, giá trị từ -128 đến 127
    - short - Số nguyên, 2 byte, giá trị từ -32768 đến 32767
    - int - Số nguyên, 4 byte, giá trị từ -2^31 đến 2^31-1
    - long: Số nguyên, 8 byte, giá trị từ -2^63 đến 2^63-1
    - float - Số thực, 4 byte
    - double - Số thực, 8 byte
    - boolean - Giá trị là true và false
    - char - ký tử Unicode, 2 byte
  - Kiểu tham chiếu (Reference)
    - String
    - Array
    - Class
    - . . .

1. Tổng quan Java(2)

- Các loại toán tử
  - Toán tử số học: + - * /
  - Toán tử một ngôi: + - ++ -- !
  - Toán tử tăng giảm: ++ --
  - Toán tử so sánh: == != > >= < <=
  - Toán tử logic: && || !
  - Java không có toán tử so sánh === vì java là kiểu dữ liệu tĩnh
- Cấu trúc điều kiện
  - Thực thi một nhóm câu lệnh dựa trên điều kiện cho trước
  - Các loại
    - IF ELSE
    - SWITCH CASE
    - Toán tử 3 ngôi

1. Tổng quan Java(3)

- Input và Outpit
  - Output: sout
    - System.out.println(): in ra có xuống dòng
    - System.out.print(): in ra không xuống dòng
    - System.out.printf(): in ra có định dạng
  - Input: scanner
    - scanner.nextLine(): Nhập chuỗi (bao gồm của khoảng trắng)
    - scanner.nextInt(): Nhập số nguyên
    - scanner.nextDouble(): nhập số thực

2. Vòng lặp và mảng

- Mảng
  - Khái niệm
    - Là một biến đặc biệt lưu được nhiều giá trị
    - Những giá trị trong mảng được sắp xếp liền kề nhau trong ô nhớ
  - Mục đích sử dụng
    - Giảm độ dài code
    - Dễ dàng quản lý danh sách
    - Duyệt dễ dàng bằng vòng lặp
  - Mảng trong java
    - Có độ dài cố định và khai báo độ dài trước khi sử dụng
    - Các phần tử trong mảng phải có cùng kiểu dữ liệu, luôn phải khai báo kiểu dữ liệu khi khởi tạo mảng
    - Không có phương thức hỗ trợ cho mảng
  - Khai báo và khởi tạo
    - Cú pháp:dataType [] arrayName = new dataType[arraySize]
      - dataType: kiểu dữ liệu
      - arrayName: tên biến mảng
      - arraySize: độ dài của mảng
  - Các cách duyệt mảng
    - Vòng lặp for
      - int[] arr = {1,2,3,4,5}for(int i = 0; i < arr.length; i++){ //code}
      - Có thể trả về giá trị và vị trí index của phần tử
      - Duyệt theo bất kỳ chiều nào và vị trí nào
      - Có thể thay đổi phần tử trong mảng
    - Vòng lặp for each
      - int[] arr = {1,2,3,4,5}for(int num: arr){ //code}
      - Chỉ trả về giá trị của phần tử
      - Chỉ có thể duyệt từ 0 đến cuối mảng
      - Không làm thay đổi phần tử trong mảng

2. Vòng lặp và mảng(1)

- Vòng lặp
  - Khái niệm
    - Thực hiện lặp đi lặp lại một đoạn code dựa trên điều kiện cho trước
  - Các loại vòng lặp
    - For
      - Sử dụng khi xác định được số lần cần lặp
      - Kiểm tra điều kiện trước khi lặp
    - While
      - Sử dụng khi chưa xác định được số lần lặp
      - Kiểm tra điều kiện trước khi lặp
    - Do while
      - Sử dụng khi chưa xác định được số lần cần lặp'
      - Thực hiện lặp ít nhất một lần rồi mới kiểm tra điều kiện
  - Câu lệnh nhảy
    - Break
      - Khi vòng lặp gặp break thì lập tức thoát khỏi vòng lặp
    - Continue
      - Khi vòng lặp gặp continue thì bỏ qua khối lệnh ở dưới và bắt đầu lần lặp tiếp theo

3. Phương thức (method)

- Khái niệm
  - Là một nhóm các câu lệnh thực hiện một nhiệm vụ cụ thể
  - Mục đích sử dụng
    - Tái sử dụng mã nguồn
    - Giúp chương trình ngắn gọn
    - Dễ đọc và dễ bảo trì
- Cú pháp
  - Khởi tạo
    - modifier returnValueType methodName (list of parameter){ //MethodBody}
  - Gọi
    - methodName(list of argument);
  - Chú thích
    - modifier: có thể là các từ khóa để duy định các tính chất khác nhau của phương thức
    - returnValueType: là kiểu dữ liệu trả về của phương thức
    - methodName: tên gọi của phương thức
    - methodBody: phần thân phương thức
    - list of parameter: danh sách các tham số của phương thức
    - list of argument: danh sách các đối số tuyền vào khi gọi hàm

3. Phương thức (method)(1)

- Các loại phương thức
  - Phân loại theo giá trị trả về
    - phương thức có return
    - Phương thức không có return
  - Phân loại theo tham số
    - Phương thức có tham số truyền vào
    - Phương thức không có tham số truyền vào
- Điều kiện để tạo nhiều phương thức cùng tên trong cùng 1 class
  - Khác nhau về số lượng tham số
  - Khác nhau về kiểu dữ liệu của tham số
  - Khác nhau về thứ tự của kiểu dữ liệu tham số

3. Phương thức (method)(2)

- Bộ nhớ Heap và Stack
  - Khái niệm
    - là một phần của bộ nhớ được JVM sử dụng để chạy chương trình java
    - JVM chia bộ nhớ được cấp từ hệ điều hành thành 2 phần Heap và Stack cho việc quản lý
  - Heap
    - Là bộ nhớ được sử dụng để lưu các object được tạo bằng từ khóa new
    - Được Garbage Collector dọn dẹp khi không có biến tham chiếu đến object trong heap
    - Dung lượng tùy thuộc vào object sử dụng
    - Tốc độ chậm hơn stack
    - Phạm vi: cho đến khi không có tham chiếu
  - Stack
    - Là bộ nhớ để lưu các biến cục bộ trong hàm và primitive
    - Tự động đóng khi hàm kết thúc
    - Dung lượng nhỏ
    - Tốc độ nhanh
    - Có phạm vi trong phương thức

3. Phương thức (method)(3)

- Tham trị và tham chiếu
  - Tham trị (pass by value)
    - là một bản sao của giá trị biến truyền vào phương thức
    - Các thay đổi với tham số bên trong phương thức không làm thay đổi giá trị gốc của biến
  - Tham chiếu (pass by reference)
    - là tham chiếu trực tiếp đến địa chỉ vùng nhớ của biến
    - Các thay đối giá trị của biến bên trong hàm sẽ ảnh hướng đến giá trị gốc của biến

4. Class và Object

- OOP
  - Khái niệm
    - Lập trình hướng đối tượng (Object Oriented Programming) là ánh xạ (đưa) các đối tượng ở thực tế vào trong lập trình]
    - Mọi thứ trong code đều là đối tượng và các đối tượng có sự tương tác với nhau
  - Ưu và nhược điểm so với POP
    - Ưu
      - Giúp tái sử dụng mã nguồn
      - Dễ bảo trì và mở rộng
      - Có tính bảo mật cao
      - Hỗ trợ mô hình thực tế
      - Tính trừ tượng và đóng gói
    - Nhược
      - Độ phức tạp cao
      - Hiệu xuất chậm hơn
      - Đòi hỏi kiến thức nền tảng vững

4. Class và Object(1)

- 4 tính chất OOP
  - Tính đóng gói (Encapsulation)
    - Che dấu thuộc tính bên trong của đối tượng (chỉ cho phép truy cập thông qua 2 phương thức setter/getter) => bảo mật đối tượng
  - Tính kế thừ (Inheritance)
    - Lớp con có thể kế thừa các đặc điểm và hành vi của lớp cha, lớp con có thể mở rộng thêm những đặc điểm và hành vi mới
  - Tính đa hình (Polymorphism)
    - Cùng một đối tượng hoặc hành vi sẽ có cách thể hiện và triển khai khác nhau tùy thuộc vào ngữ cảnh
  - Tính trừu tượng (Abstraction
    - Bỏ qua những thuộc tính và hành vi không cần thiết, chi quan tâm đến những thuộc tính và hành vi cần giải quyết của đối tượng
    - Chỉ quan tâm đầu vào và đầu ra, không quan tâm đến tiến trình triển khai bên trong

4. Class và Object(2)

- Phân biệt
  - Class
    - Là một khuôn mẫu chung quy định các đặc điểm và hành vi của một nhóm các đối tượng
  - Object
    - Là một thể hiện của class, có các đặc điểm và hành vi cụ thể
- this
  - Được sử dụng để đại diện cho đối tượng hiên tại
  - Có thể sử dụng this để truy cập đến các thành phần của đối tượng hiện tại
- Constructor
  - Khái niệm
    - là một phương thức đặc biệt giúp khởi tạo đối tượng của một lớp.
    - Constructor tự động gọi khi đối tượng mới được tạo ra bằng từ khóa "new"
  - Đặc điểm
    - sử dụng new để gọi constructor
    - trùng tên với tên lớp
    - không có kiểu dữ liệu trả về
    - Trả về một đối tượng
    - một lớp có thể có nhiều Constructor
    - Sử dụng this để gọi constructor khác
    - có constructor mặc định không tham số
  - So sanh với method
    - Constructor
      - Tên phải trùng với tên của class
      - Mục đích giúp khởi tạo object
      - Không có dữ liệu trả về
      - Tự động gọi khi tạo đối tượng
    - Method
      - Tên được người dùng đặt bất kỳ
      - Mục đích giúp xử lý các hành vi của object
      - Có kiểu dữ liệu trả về
      - Được gọi thủ công qua object

5. Access modifier và static

- Access modifier
  - Khái niệm
    - Là từ khóa được sử dụng để quy định mức độ truy cập đến lớp và thành phần của lớp
  - Các loại AM
    - Public
      - Có thể truy cập từ bất cứ đâu
    - Protected
      - Các phương thức và thuộc tính được phép truy cập trong cùng một package và ở các lớp con (kế thừa)
    - Default (Non-access modifier)
      - Nếu không có AM thì mức này được áp dụng. Lớp và các thành phần của lớp được truy cập ở những nơi trong cùng package
    - Private
      - Chỉ được phép truy cập trong cùng một lớp

5. Access modifier và static(1)

- Static
  - Khái niệm
    - Được sử dụng để khai báo thuộc tính và phương thức của lớp (khác với của đối tượng)
    - Các phần phần static trực thuộc lớp thay vì đối tượng
    - Có thể truy xuất các thành phần static bằng cách sử dụng lớp
    - Không cần khai báo đối tượng vẫn có thể sử dụng các thành phần static
  - Được sử dụng
    - trước thuộc tính (biến static)
      - Có thể làm thuộc tính chung, dùng chung dữ liệu với các object hoặc instance của lớp
      - có thể truy cập trực tiếp thông qua tên lớp
    - trước method (phương thức static)
      - Có thể truy cập biên static để thay đổi giái trị
      - Không thể được ghi đè
      - Sử dụng làm các phương thức tiện ích
    - trước block (khối static)
      - Dùng để thay đổi giá trị của biến static
      - Được thực thi trước hàm main tại thời điểm tải lớp
      - Một lớp có thể có nhiều khối static
    - trước lớp (lớp static)
      - Là lớp static chỉ khi nằm trong lớp khác
      - Có thể được truy cập mà không cần lớp bên ngoài
  - Ràng buộc
    - static chỉ có thể làm việc TRỰC TIẾP với một static
    - Không dùng được với từ khóa this và super

5. Access modifier và static(2)

- Các biến trong java
  - Biến cục bộ (Local)
    - Được khai báo trong phương thức, constructor hoặc block
    - Cần tạo giá trị mặc định mới có thể sử dụng
    - Không dùng AM để khai báo
    - Dùng với method
    - => Lưu ở vùng nhớ Stack
  - Biến toàn cục (Instance)
    - Được khai báo bên ngoài các phương thức, constructor, block
    - Được tạo ra khi đối tượng mới được tạo bằng từ khóa new
    - Dùng với Object
    - Được phép sử dụng AM để khai báo
    - => Lưu ở vùng nhớ Heap
  - Biến static
    - Được khai báo trong một class với từ khóa static, phía bên ngoài các phương thức
    - Sẽ chỉ có duy nhất một bản sao được tạo ra
    - Được truy cập thông qua tên class chứa nó
    - Chỉ được gọi trong các phương thức static
    - Sử dụng để tạo các phương thức util, để sử dụng nhiều lần
    - Khối static luôn chạy trước hàm main
    - => Lưu ở vùng nhớ Method Area

5. Access modifier và static(3)

- Tính bao đóng(Encapsulation)
  - Khái niệm
    - Là cơ chế bảo mật thuộc tính bên trong của đối tượng, không cho phép truy cập trực tiếp mà phải truy cập thông qua phương thức getter và setter
  - Mục đich
    - Kiểm soát việc truy cập và thay đổi thuộc tính
    - Tăng tính bảo mật và an toàn
  - Cách sử dụng
    - Khai báo thuộc tính của lớp là private
    - Tạo các getter và setter là public

6. Kế thừa

- Khái niệm
  - (Inheritance)
  - Lớp con có thể kế thừa các thuộc tính và hành vi của lớp cha, lớp con có thể mở rộng các hành vi và thuộc tính riêng
  - Mục đích
    - Tái sử dụng code
    - Dễ dàng bảo trì
    - Mở rộng tính năng
    - Tăng tính linh hoạt
- Đặc điểm của kế thừa
  - Sử dụng extends để triển khai
  - Lớp con là subclass, lớp cha là super class
  - Thành phần private không thể kế thừa
  - Constructor không thể kế thừa mà chỉ được gọi bằng super
  - Lớp con có thể mở rộng thuộc tính và phương thức mới
  - Java không hỗ trợ đa kế thừa
    - vì muốn tránh sự mơ hồ và xung đột khi một lớp kế thừa từ nhiều lớp cha -> diamond problem

6. Kế thừa(1)

- Final
  - biến final
    - Hằng số
  - phương thức final
    - Ngăn chặn ghi đè
  - Lớp final
    - Ngăn chặn kế thừa
- Super
  - Là biến tham chiếu trực tiếp đến đối tượng của lớp cha gần nhất
  - Sử dụng
    - Gọi instance của lớp cha
    - Gọi phương thức từ lớp cha
    - Gọi constructor từ lớp cha

6. Kế thừa(2)

- Lớp Object
  - Là lớp gốc của tất cả các lớp trong java
  - Tất cả các lớp trong java đều kế thừa lớp Object
  - Phương thức
    - toString()
      - Trả về chuỗi ký tự đại diện cho đối tượng
    - getClass()
      - Trả về đối tượng lớp class của đối tượng
    - hashCode()
      - Trả về hash code của đối tượng
    - equals()
      - So sánh 2 đối tương (==)
    - . . . . .

- **Tính đa hình(Polymorphism)**
  - Khái niệm
    - Cùng một lớp có nhiều cách thể hiện khác nhau tùy thuộc vào ngữ cảnh
    - Cùng một phương thức, có nhiều cách triển khai khác nhau tùy thuộc vào lớp
  - Thể hiện
    - Thể hiện qua thời điểm biên dịch (đa hình compile) -> Overloading
    - Thể hiện tại thời điểm chạy chương trình (đa hình runtime) -> Overriding

- **Overriding**
  - Cho phép lớp con định nghĩa lại phương thức của lớp cha
  - Điều kiện
    - Cùng tên, cùng tham số, cùng kiểu giá trị trả về
    - Access modifier phải bằng hoặc cao hơn
    - Có thể sử dụng @Override để đánh dấu
- **Overloading**
- Cho phép lớp có nhiều phương thức cùng tên
  - Điều kiện
    - Cùng tên, khác tham số
    - Kiểu giá trị trả về tuỳ chọn
    - Access modifier phải bằng hoặc cao hơn

6. Kế thừa(4)

- Ép kiểu
  - Là quá trình chuyển đổi kiểu dữ liệu của một biến sang kiểu dữ liệu khác
  - 2 loại
    - Ngầm định (Implicit casting)
      - Kiểu nhỏ -> kiểu lớn
    - Tường minh (explicit casting)
      - Kiểu lớn -> kiểu nhỏ
  - Lưu ý
    - Phải trong mối quan hệ kế thừa
      - Lỗi compile
    - Phải Implicit thì mới có Explicit
      - Lỗi runtime
    - Cần kiểu tra kiểu dữ liệu trả về trước khi ép kiểu
      - instanceof

7. Abstract class và Interface

- Tính trừu tượng
  - Abstraction
  - Khái niệm
    - Bỏ qua những đặc điểm và hành vi không cần thiết, chỉ quan tâm đến những đặc điểm hành vi cần giải quyết
    - Chỉ quan tâm đầu vào đầu ra, không quan tâm đến tiến trình triển khai bên trong
  - Thể hiện
    - Abstract class
    - Interface
- Abstract method
  - Là phương thức được khai báo không có phần thân
  - Đặc điểm
    - Không có phần thân
    - Chỉ tồn tại trong lớp abstract hoặc interface
    - Lớp con buộc phải override
    - Không thể là final
    - Giúp định nghĩa hành vi chung

7. Abstract class và Interface(1)

- Abstract class
  - Là lớp trừu tượng, không thể tạo đối tượng
  - Đặc điểm
    - Không thể khởi tạo đối tượng
    - Không thể dùng từ khóa final
    - Có thuộc tính và method bình thường
    - 1 lớp abstract có thể không có phương thức abstract
    - 1 lớp có một phương thức abstract thì đó là lớp abstract
    - 1 lớp kế thừa 1 lớp abstract thì lớp phải triển trai tất cả phương thức abstract
    - 1 lớp abstract khi kế thừa 1 lớp abstract thì không nhất thiết phải triển khai các phương thức abstract
  - Sử dụng
    - Một lớp cụ thể extends một lớp abstract
      - Phải triển khai tất cả phương thức abstract của lớp cha
    - Một lớp abstract extends một lớp abstract khác
      - Không cần phải triển khai phương thức abstract

7. Abstract class và Interface(2)

- Interface
  - Là bản thiết kế, quy định hành vi chung của các class sử dụng
  - Đặc điểm
    - Không có constructor, không thể khởi tạo đối tượng
    - Các phương thức mặc định là public abstract
    - Biến trong interface mặc định là public static final
    - Từ Java 8, có thể tạo các phương thức có phần thân (default, static)
    - 1 lớp cụ thể khi implements một interface
      - Phải override toàn bộ phương thức abstract
    - 1 lớp abstract khi implemets interface
      - Không cần thiết triển khai phương thức abstract
    - 1 class có thể implements nhiều interface
    - 1 interface có thể extends nhiều interface

7. Abstract class và Interface(3)

- Comparable và Comparator
  - Dùng để so sánh 2 đối tượng
    - bằng 0
      - bằng nhau
    - lớn hơn 0
      - Tăng dần
    - bé hơn 0
      - giảm dần
  - Comparable
    - Implements giao tiếp cho đối tượng so sánh
    - Phương thức compareTo(o)
    - Cú pháp Collections.sort(list)
    - Sử dụng khi class chỉ cần một tiêu chí sắp xếp
  - Comparator
    - Không cần implements giao tiếp cho đối tượng so sánh
    - Phương thức compare(o1, o2)
    - Cú pháp Collections.sort(list, Comparator)
    - Sử dụng khi muốn linh hoạt nhiều kiểu sắp xếp hoặc không dùng class

8. Clean code

- Clean code
  - là chỉ nhưng mã nguồn tốt (mã sạch)
  - Đặc điểm
    - Dễ đọc
    - Rõ ràng
    - Đơn giản
    - Dễ bảo trì
    - Ít trùng lặp
    - Được tổ chức tốt
- Smell code
  - Là chỉ những mã nguồn gây khó khăn cho việc đọc, bảo trì và mở rộng
  - Đặc điểm
    - Đặt tên sai
    - Phương thức hoặc lớp quá dài
    - Phương thức hoặc lớp xử lý quá nhiều việc
    - Lạm dụng comment
    - Code bị lặp lại
    - Sử dụng giá trị magic

8. Clean code(1)

- Refactor
  - Là chỉnh sửa mã nguồn sạch hơn mà không làm thay đổi hành vi của hệ thống
  - Có thể đổi tên biến, tách hàm, tối ưu hóa cấu trúc
- Coding convention
  - Là quy tắc khi viết code
  - MVC
    - Model
      - Biểu diễn dữ liệu
      - Xử lý nghiệp vụ
      - Kết nối DB
    - View
      - Hiển thị
      - Tương tác với người dùng
    - Controller
      - Điều hướng
    - C -> S -> R

9. Kiểm thử

- Khái niệm
  - là nghiên cứu được thực hiện để khẳng định chất lượng của sản phầm phần mềm
  - Lí do
    - Nâng cao chất lượng sản phẩm
    - Tìm ra lỗi bug. hạn chế rủi ro
    - Cung cấp thông tin cho các bên liên quan
    - Đảm bảo chương trình chạy đúng yêu cầu
- 4 mức kiểm thứ
  - Unit Testing
    - Kiểm thử đơn vị
  - Intergration Testing
    - Kiểm thử tích hợp
  - System Testing
    - Kiểm thử hệ thống
  - User Testing / Acceptance Testing
    - Kiểm thử người dùng / hệ thống

9. Kiểm thử(1)

- TDD
  - Test Driven Development
  - Phương pháp kiểm thử cải tiến cho nhà pháp triển chuyên nghiệp, nâng cao năng suất phần mềm
  - 2 thành phần
    - TFD
      - Test-First Development
    - Refactoring
  - Các bước thực thi
    - Tạo mới/sử kiểm thử <-> Chạy kiểm thử -> Viết mã nguồn cho sản phẩm <-> Chạy tất cả các kiểm thử -> Dọn dẹp mã nguồn -> (lặp lại)
- Unit Testing
  - Là một framework dùng cho java, được tích hợp sẵn cùng IDE. Dùng để kiểm thử mức đơn vị của đoạn mã nhỏ

10. DSA: Danh sách

- Java Collection Framework
  - Là khuôn khổ cung cấp các class và interface dùng để lưu trữ và thao tác với một nhóm đối tượng dữ liệu
  - Thao tác chính: CRUD
  - Các interface chính
    - List
      - Cấu trúc dữ liệu tuyến tính, cho phép phần tử trùng lặp
    - Set
      - Cấu trúc dữ liệu phi tuyến tính, không cho phép trùng lặp
    - Queue
      - Cấu trúc dữ liệu theo quy tắc hàng đợi FIFO
    - Map
      - Cấu trúc dữ liệu lưu trữ theo các cặp <key-value>

10. DSA: Danh sách(1)

- So sánh List và Set
  - List
    - Là một danh sách có thứ tự
    - Cho phép phần tử trùng lặp
    - Có thể lặp lại phần tử theo thứ tự
    - Có thể chứa nhiều phần tử null
    - Các lớp triển khai
      - ArrayList
        - Là lớp triển khai từ List dựa trên cơ chế mảng động
      - LinkedList
        - Là lớp triển khai từ List dựa trên cơ chế liên kết node
      - Vector
        - lưu trữ phần tử trong mảng có đồng bộ hóa
      - Stack
        - Kế thừa từ Vector, hoạt động theo LIFO
  - Set
    - Là chuỗi không có thứ tự
    - Không cho phép trùng lặp
    - Không đảm bảo thứ tự khi lặp
    - Chỉ cho phép 1 phần tử null
    - Các lớp triển khai
      - HashSet
      - LinkedHashSet
      - TreeSet

10. DSA: Danh sách(2)

- So sánh Array và ArrayList
  - Array
    - Có kích thước cố định
    - Lưu được kiểu dữ liệu nguyên thủy và đối tượng
    - Không có các phương thức hỗ trợ
    - Nhanh hơn
    - Không hỗ trợ kiểu generic
  - ArrayList
    - Kích thước động
    - Chỉ lưu được kiểu dữ liệu đối tượng
    - Có phương thức hỗ trợ
    - Chậm hơn
    - Hỗ trợ kiểu Generic

10. DSA: Danh sách(3)

- So sánh ArrayList và LinkedList
  - ArrayList
    - Sử dụng mảng động để lưu phần tử
    - Thao tác thêm, xóa chậm hơn
    - Truy xuất phần tử nhanh
  - LinkedList
    - Sử dụng các node liên kết để lưu trữ
    - Thao tác thêm, xóa nhanh
    - Truy xuất phần tử chậm hơn

11. DSA: Stack và Queue

- Generic
  - Khái niệm
    - Là cơ chế cho phép sử dụng kiểu dữ liệu như là tham số cho các interface, method và class
    - Có thể định nghĩa class, method với kiểu dữ liệu generic, sau đó compiler sẽ thay thế bằng một kiểu dữ liệu cụ thể
  - Ưu điểm
    - Tạo ra các interface, method dùng chung => tái sử dung code
    - Phát hiện lỗi tại thời điểm biên dịch
    - Không cần ép kiểu
  - Nhược điểm
    - Không thể dùng với kiểu dữ liệu nguyên thủy
    - Không thể khởi tạo đối tượng trực tiếp
    - Không thể dùng từ khóa static

11. DSA: Stack và Queue(1)

- Stack
  - Là cấu trúc dữ liệu hoạt động theo quy tắc FILO/LIFO
  - Một số method
    - push()
      - Thêm phần tử mới vào cuối
    - pop()
      - Lấy và xóa phần tử cuối
    - peek()
      - Lấy nhưng không xóa phần tử cuối
    - isEmpty()
      - Kiểu tra xem stack có trống hay không
    - size()
      - Lấy kích thước hiên tại

11. DSA: Stack và Queue(2)

- Queue
  - Là cấu trúc dữ liệu hoạt động theo quy tắc FIFO
  - Một số method
    - add()/offer()
      - Thêm phần tử mới vào cuối'
    - remove()
      - Lấy và xóa phần tử đầu
      - trả exception nếu danh sách trống
    - poll()
      - Lấy và xóa phần tử đầu
      - Trả về null nếu danh sách trống
    - element()
      - Lấy nhưng không xóa phần tử đầu
      - Trả exception nếu danh sách trống
    - peek()
      - Lấy nhưng không xóa phần tử đầu
      - Trả về null nếu danh sách trống
    - isEmpty()
      - Kiểm tra danh sách có trống hay không
    - size()
      - Trả về kích thước hiện tại
  - Các class triển khai
    - LinkedList
      - Danh sách lưu trữ hoạt động theo liên kết node
    - PriorityQueue
      - Hàng chờ ưu tiên, sắp xếp theo tự nhiên hoặc comparator, không đảm bảo FIFO
    - Dequeue
      - ArrayDequeue
        - Hàng đợi 2 đầu hiệu xuất cao

12. DSA: Set và Map

- Set
  - Set là cấu trúc dữ liệu lưu trữ các phần tử không tuyến tính (không có trật tự) và không cho phép trùng lặp
  - Các lớp triển khai
    - HashSet
      - mang đặc điểm của Set
    - LinkedHashSet
      - Các phần tử thêm vào có trật tự
    - TreeSet
      - Các phần tử được sắp xếp (tăng dần hoặc giảm dần)
- Map
  - Map là cấu trúc dữ liệu lưu trữ các phần tử theo cặp <Key, Value>
  - Đặc điểm
    - Mỗi một Key tương ứng với một Value
    - Key không được trùng lặp
    - Mỗi cặp <Key, Value> được gọi là một entry
  - Các lớp triển khai
    - HashMap
      - Các phần tử entry không có trật tự
    - LinkedHashMap
      - Các phần tử entry thêm vào có trật tự
    - TreeMap
      - Các phần tử entry được sắp xếp theo key

12. DSA: Set và Map(1)

- Tree
  - Lưu trữ dữ liệu trên các node, các node có mối quan hệ cha con, node trên cùng được gọi là node gốc
  - Binary Tree
    - Mỗi node có 0-2 node con, left-subtree và right-subtree
    - Binary Search Tree (BST)
      - Biểu diễn bằng một tập các node liên kết với nhau, mỗi node chứa 2 liên kêt: liên kết node trái và liên kết node phải
      - Duyệt
        - Inorder
- Trái -> Cha -> Phải
        - Preorder
- Cha -> Trái -> Phải
        - Postorder
- Trái -> Phải -> Cha
        - Breath-first
- Duyệt lần lượt theo từng level

13. Search algorithms

- 2 loại phổ biến
  - linear search
    - Tìm kiếm tuyến tính
    - thường được thực hiện với mảnh hay danh sách nhỏ, chưa được sắp xếp thứ tự
  - binary search
    - Tìm kiếm nhị phân
    - thường được thực hiện với mảng hay danh sách đã được sắp xếp
- Độ phức tạp thuật toán
  - là để tính chương trình chạy nhanh hay chậm
    - Dựa vào
      - Thuật toán
      - Máy tính
  - Quy tắc
    - O(1): code đơn giản
    - O(n): vòng lặp for
    - Nối tiếp: (max) (On/Om)
    - song song: O(n^2)

13. Search algorithms(1)

- Linear search
  - Ý tưởng
    - Duyệt mảng từ đầu đến cuối
    - so sánh giá trị với phần tử mảng
    - Nếu tìm thấy thì return index
    - Nếu không tìm thấy thì return -1
  - Độ phức tạp
    - Tốt nhất
      - Ở đầu danh sách
      - O(1)
    - Xấu nhất
      - Ở cuối danh sách
      - O(n)

13. Search algorithms(2)

- Binary search
  - Ý tưởng
    - B1: mảng được sắp xếp
    - B2: lấy phần tử giữa mảng so sánh với value
    - B3
      - Nếu value = mid thì trả về index
      - Nếu value < mid thì giá trị cần tìm ở bên trái
      - Nếu value > mid thì giá trị bên phải
      - Lặp lai B2
    - B4: Nếu số phần tử = 0 thì return -1
  - Độ phức tạp
    - Tốt nhất
      - Phần tử cần tìm nằm ở giữa
    - Xấu nhất
      - Nằm ở đầu hoặc cuối
      - O(log2 n)

14. Sort algorithms

- Mục đích
  - Giúp dễ xem, dễ tìm kiếm và quản lý một danh sách
- Bubble sort
  - Sắp xếp nổi bọt
  - Ý tưởng
    - so sánh 2 phần tử liền kề nếu không thỏa mãn điều kiện (tăng/ giảm) thì sẽ hoán đổi vị trí giữa 2 phần tử cho nhau
  - Độ phức tạp
    - Tốt nhất
      - Mảng được sắp xếp
      - O(n)
    - Xấu nhất
      - Mảng chưa sắp xếp
      - O(n^2)

14. Sort algorithms(1)

- Selection sort
  - Sắp xếp chọn
  - Ý tưởng
    - Bước 1:
      - Tìm ra phần tử nhỏ nhất trong dãy gồm n phần tử
      - Hoán đổi vị trí phần tử nhỏ nhất với đầu mảng
    - Bước 2
      - Loại bỏ phần tử đầu tiên của mảng
      - Xem dãy đang xét gồm n-1 phần tử (bắt đầu từ pt thứ 2)
      - Lặp lại bước 1 cho đến khi dãy đang xét chỉ còn một phần từ
  - Độ phức tạp
    - Tốt nhất
      - O(n^2)
    - Xấu nhất
      - O(n^2)

14. Sort algorithms(2)

- Insertion sort
  - Sắp xếp chèn
  - Ý tưởng
    - Luôn duy trì mảng con được sắp xếp ở trước mảng cần sắp xếp (ban đầu chỉ 1 phần tử)
    - Chèn một phần tử vào trong mảng con được sắp xếp, phần tử được chèn phải duy trì được mảng con được sắp xếp
  - Độ phức tạp
    - Tốt nhất
      - O(n)
    - Xấu nhất
      - O(n^2)

15. Exception

- Trong quá trình thực thi chương trinh có thể xảy ra
  - Error
    - Lỗi nghiêm trọng
    - Không thể xử lý'
    - Chương trình sẽ chết
  - Exception
    - Ít nghiêm trọng hơn
    - Có thể xử lý được
  - Lỗi logic
    - Xử lí bằng debug

15. Exception(1)

- Khái niệm
  - Là những lỗi phát sinh trong quá trình thực thi
  - 2 loại exception
    - Checked exception
      - Xảy ra tại thời điểm biên dịch
        - Cú pháp
      - Yêu cầu xử lý thì chương trình mới chạy được
    - Unchecked exception
      - Xảy ra tại thời điểm thực thi
      - Không cần xử lý vẫn chạy được
      - Có thể gây chết chương trình nếu không xử lý

15. Exception(2)

- Cách xử lý exception
  - Xử lý tại chỗ (Try-catch)
    - try
      - Để những đoạn code có nghi ngờ gây ngoại lệ thì bỏ vào try
    - catch
      - Để vào những đoạn code cần làm khi ngoại lệ xảy ra
    - Các lưu ý
      - trong một khối try có thể có nhiều khối catch
      - Các khối catch phải sắp xếp từ lớp con đến lớp cha
      - Khối finally luôn chạy dù ngoại lệ có xảy ra hay không
  - Sử dụng throws
    - Né ngoại lệ ra bên ngoài method để nơi nào sử dụng method đó sẽ xử lý

15. Exception(3)

- try-with-source
  - Là cú pháp giúp tự động đóng các tài nguyên trong khối try mà không cần gọi lệnh đóng trong finally
  - Cách sử dụng: Khởi tạo tài nguyên trong dấu () sau try
  - Điều kiện
    - Tài nguyên phải impl interface autoCloseable
    - Có thể khai báo nhiều tài nguyên cách nhau bằng dấu (;)
- Sử dụng finally
  - Khi muốn thực thi nhưng dòng code quan trọng dù exception có xảy ra hay không
    - Đóng stream
    - Đóng kết nối

16. IO: Read & write file

- Stream
  - (dòng) là hoạt động nhập/ xuất dữ liệu
  - Ví dụ: nhập dữ liệu từ bàn phím, đọc dữ liệu từ file, in dữ liệu ra màn hình,...
  - 2 loại dòng
    - Dòng vào
      - Cho phép chương trình đọc dữ liệu từ một nguồn
    - Dòng ra
      - Cho phép chương trình ghi dữ liệu lên nó để đến một đích nào đó
  - Các luồng dữ liệu
    - Dòng byte (byte-based stream)
      - Hỗ trợ việc nhập xuất dữ liệu theo byte
      - Thường dùng khi đọc ghi dữ liệu nhị phân
    - Dòng charactor (charactor-based stream)
      - Hỗ trợ việc nhập xuất dữ liệu kiểu kí tự

16. IO: Read & write file(1)

- Input
  - FileReader
    - Đọc từng kí tự
  - BufferReader
    - Đọc từng dòng
- Output
  - FileWriter
    - Ghi từng kí tự
  - BufferWriter
    - Ghi nhanh hơn, hỗ trợ newLine()

17. IO: Serialization

- Khái niệm
  - Serialization là chuyển đổi một đối tượng Object thành byte stream
  - Deserialization là chuyển đổi ngược lại một byte stream thành một Object
- Transient
  - Dùng để chặn một thuộc tính không cho serialization
- Lưu ý khi sử dụng
  - Nếu class cha đã implement Serialization thì lớp con không cần phải implement lại
  - Nếu không muốn một thuộc tính serialization thì sử dụng transienet trước thuộc tính đó
  - Biến static không được transient

17. IO: Serialization(1)

- Input
  - FileInputStream
    - Đọc dữ liệu nhị phân(byte) từ file
  - ObjectInputStream
    - Đọc (desirialize) một đối tượng từ luồng nhị phân
- Output
  - FileOutputStream
    - Ghi dữ liệu nhị phân(byte) vào file
  - ObjectOutputStream
    - Ghi (serialize) một đối tượng vào luồng nhị phân

18. Thread

- Mutil-TaskingMulti-Threading
  - Multi-Tasking
    - Là khả năng chạy đồng thời nhiều chương trình cùng lúc trên hệ điều hành
  - Multi-Threading
    - Là khả năng thực hiện đồng thời nhiều tiểu trình trong một chương trình
- Khái niệm
  - Là đơn vị nhỏ nhất của mã thực thi mà đoạn mã đó thực hiện một nhiệm vụ cụ thể
- Các cách tạo Thread
  - Kế thừa từ lớp Thread có sẵn
    - Bước 1: Tạo một lớp mới kế thừa từ lớp Thread
    - Bước 2: Override phương thức run()
    - Bước 3: Tạo đối tượng Thread và gọi phương thức start()
  - Thực thi interface Runnable có sẵn
    - Bước 1: Tạo một lớp mới implement interface Runnable
    - Bước 2: Thực thi phương thức run()
    - Bước 3: tạo đối tượng thread và gọi phương thức start

18. Thread(1)

- Các trạng thái
  - New
    - Một thread tạo ra chưa được gọi start
  - Ready
    - thread ở trạng thái sẵn sàng chờ start() được gọi
  - Running
    - thread đang ở trạng thái làm việc
  - Sleeping
    - thread dừng tạm thời, sau đó có thể tiếp tục hoạt động
  - Waiting
    - thread vào trạng thái chờ, sử dụng khi 2 thread trở lên đồng thời hoạt động
  - Blocked
    - thread và trạng thái bị chặn khi đang đợi một sự kiện nào đó
  - Dead
    - thread ngừng hoạt động, sau khi thực hiện run() hoặc gọi stop()

18. Thread(2)

- Các phương thức
  - start()
    - bắt đầu thực hiện một thread, JVM gọi phương thức run() trên thread
  - sleep(long ms)
    - làm thread ngừng hoạt động dựa theo số miliseconds
  - join()
    - Đợi cho mọt thread khác chết
  - join(long ms)
    - đợi cho một thread chết dựa theo số miliseconds
- Đồng bộ, bất đồng bộ'
  - sycnhronous
    - Là khả năng kiểm soát truy cập của nhiều luồng đến bất kỳ nguồn tài nguyên dùng chung nào
  - asynchronous
    - Cho phép truy cập không theo tuần tự. Các đoạn code ở dưới có thể chạy trước các đoạn code ở trên, không cần phải đợi lẫn nhau

19. String và Regex

- String (chuỗi)
  - Dùng để lưu trữ chuỗi và văn bản
  - Là bất biến (immutable) => không thể thay đổi giá trị
  - 2 cách khởi tạo
    - C1: String s = "abc"
      - String Literal
      - Lưu ở trong String Pool
    - C2: String s = new String("abc")
      - Lưu ở trong Heap
- String Pool
  - Là một vùng nhớ đặc biệt nằm trong vòng nhớ Heap, dùng để lưu trữ các biến được khai báo theo kiểu String
  - Giúp quản lý giá trị String trong vùng nhớ Heap tối ưu hơn

19. String và Regex(1)

- StringBuffer
  - có thể thay đổi (mutable)
  - Có đồng bộ
- StringBuilder
  - Có thể thay đổi (mutable)
  - Không có đồng bộ
- Regex
  - Regular Expression - Biểu thức chính quy
  - Là một chuỗi mẫu được sử dụng để quy định dạng thức của các chuỗi
  - Ứng dụng
    - Tìm kiếm
    - Thay thế
    - Xác thực
    - Tách dữ liệu

20. SOLID

- Khái niệm
  - Là viết tắt của 5 nguyên tắc thiết kế hướng đối tượng cơ bản
  - S - Single-responsibility Principle
    - Nguyên lý trách nhiệm duy nhất
  - O - Open-closed Principle
    - Nguyên lý đóng mở
  - L - Liskov Substitution Priciple
    - Nguyên lý thay thế Liskov
  - I - Interface Segregation Principle
    - Nguyên lý phân tách interface
  - D - Dependency Inversion Principle
    - Nguyên lý đảo ngược phụ thuộc

20. SOLID(1)

- Single-responsibility
  - Nguyên lý trách nhiệm duy nhất
  - Một class chỉ nên thực hiện một nhiệm vụ duy nhất
  - Mục đích: dễ bảo trì, quản lý, mở rông
- Open-closed
  - Nguyên lý đóng mở
  - Có thể mở rộng chức năng nhưng không làm thay đổi mã nguồn
  - Mục đích: mở rộng
- Liskov Substitution
  - Nguyên lý thay thế Liskov
  - Các đổi tượng lớp con có thể thay thế được cho lớp cha mà không làm thay đổi tính đúng đắn của chương trình
  - Mục đích : tránh sai sót khi mở rộng

20. SOLID(2)

- Interface Segregation
  - Nguyên lý phân tách interface
  - Không nên tạo một interface gồm nhiều phương thức mà nên chia nhỏ các interface với mục đích cụ thể
  - Mục đích: tránh dư thừa mã nguồn
- Dependency Inversion
  - Nguyên lý đảo ngược phụ thuộc
  - Các module cấp cao không nên phục vào module cấp thấp mà cả 2 nên phụ thuộc vào cái trừu tượng
  - Cái trừu tượng không nên phụ thuộc vào cái cụ thể
  - Mục đích: giảm sự phụ thuộc