1. **Introduction to Java**

|  |  |
| --- | --- |
| Java là gì? | Java là ngôn ngữ lập trình hướng đối tượng.  Write once, run anywhere (viết một lần, chạy ở mọi nơi):  chỉ cần viết mã một lần cho một ứng dụng và sau đó có thể chạy ứng dụng đó trên nhiều nền tảng khác nhau mà không cần phải viết mã lại từ đầu cho mỗi nền tảng. |
| JDK, JRE, JVM | JDK (Java Development Kit): là tool- bộ công cụ phát triển ứng dụng bằng ngôn ngữ Java.  JRE (Java Runtime Environment): môi trường thực thi ứng dụng Java.  JVM (Java Virtual Machine): máy ảo Java, thực thi mã bytecode (.class) sang mã máy. |
| Biên dịch, thông dịch | Biên dịch: dịch toàn bộ file 1 lần, sau đó sử dụng kết quả biên dịch mà không cần dịch lại.  Thông dịch: dịch từ dòng lệnh, từ file .java sang file .class, từ file .class được JVM thông dịch ra mã máy. |
| Các nền tảng Java | Có 4 nền tảng :   * Java Platform, Standard Edition (Java SE) * Java Platform, Enterprise Edition (Java EE) * Java Platform, Micro Edition (Java ME) * Java FX |
| Kiểu dữ liệu và khai báo biến hằng | Java có 2 kiểu dữ liệu:   * Kiểu nguyên thủy:  1. byte: 8-bit, giới hạn từ -128 đến 127. MĐ: 0 2. short:16-bit, giới hạn từ -32,768 đến 32,767. MĐ: 0 3. int: 32-bit, giới hạn từ -2^31 đến 2^31 - 1. MĐ: 0 4. long: 64-bit, giới hạn từ -2^63 đến 2^63 - 1. MĐ: 0L 5. float: 32-bit, số thực. MĐ: 0.0f 6. double: 64-bit, số thực. MĐ: 0.0d 7. boolean: true và false. MĐ: false 8. char: 16-bit, lưu ký tự Unicode. MĐ : ‘\u0000’  * Kiểu tham chiếu:  1. String: MĐ: null 2. Array 3. Object   Khai báo biến hằng: dùng từ khóa final. |
| Toán tử | Các toán tử cơ bản:   * Toán tử gán (assignment) * Toán tử số học (arithmetic) * Toán tử so sánh (comparison) * Toán tử logic |
| Câu lệnh điều kiện | So sánh if và swich-case:   |  |  | | --- | --- | | if | swich-case | | Có thể sử dụng để so sánh lớn hơn, nhỏ hơn… | Chỉ có thể sử dụng để so sánh bằng hoặc khác nhau | | Mỗi câu lệnh if có một biểu thức  điều kiện, với giá trị trả về là true  hoặc false | Tất cả các trường hợp (case) đều so sánh với giá trị của biểu thức điều kiện duy nhất | | Biểu thức điều kiện cần trả về giá trị kiểu boolean | Biểu thức điều kiện cần trả về giá trị là kiểu byte, short, char, int, hoặc String | | Chỉ có một khối lệnh được thực thi nếu điều kiện đúng | Nếu điều kiện đúng mà không có câu lệnh break thì tất cả các khối lệnh ở phía sau cũng được thực thi | |

**2. Loop & Array**

|  |  |
| --- | --- |
| Mảng trong Java là gì ? | Mảng là tập hợp các phần tử có cùng kiểu dữ liệu và lưu gần nhau trong ô nhớ. |
| Khai báo và khởi tạo mảng | Có 2 cách khai báo:   * elementType[ ] arrayName; //Nên dùng cách này * elementType arrayName[ ]; //Không nên   Có 2 cách khởi tạo mảng:   * Dùng từ khóa new: arrayRefVar = new elementType[arraySize]; * Khởi tạo nhanh: elementType[ ] arrayRefVar = {value0, value1, ..., valuek}; |
| So sánh mảng trong JavaScript và Java | |  |  | | --- | --- | | JavaScript | Java | | Kích thước có thể thay đổi | Kích thước cố định | | Lưu được nhiều giá trị có các kiểu dữ liệu khác nhau | Các phần tử phải cùng kiểu dữ liệu | | Giá trị mặc định các phần tử là underfined | Giá trị mặc định của phần tử phụ thuộc vào kiểu dữ liệu khai báo của mảng | | Có nhiều phương thức hỗ trợ | Chỉ có .lenght | |
| Vòng lặp | Vòng lặp for-each:  for (dataType variableName : myArray){  …  } |

**3. Method**

|  |  |
| --- | --- |
| Method? | Là nhóm các câu lệnh thực thi một nhiệm vụ nhất định nào đó.  Mục đích: tái sử dụng mã nguồn. |
| Khai báo phương thức | Cú pháp:  modifier returnValueType methodName(listof parameters) {  // *Method body;*  }  Trong đó:  • modifier có thể là các từ khoá để quy định các tính chất khác nhau của phương thức  • returnValueType là kiểu dữ liệu trả về của phương thức  • methodName là tên gọi của phương thức  • listofparameters là danh sách các tham số của phương thức, có thể có hoặc không.  • Methodbody là phần thân của phương thức |
| Kiểu dữ liệu trả về | Một phương thức có thể trả về một giá trị , bắt buộc phải có từ khóa return trong phần body phương thức và phải cùng kiểu dữ liệu khai báo ở đầu method. Nếu phương thức không trả về giá trị thì có kiểu dữ liệu trả về void |
| Phương thức main() | Phương thức main() là một phương thức đặc biệt trong Java  Phương thức main() là điểm khởi đầu (entrypoint) cho một chương  trình  Phương thức main() được gọi bởi JVM  Header của phương thức main() được quy định sẵn |

**4. Class & Object**

|  |  |
| --- | --- |
| OOP là gì? | OOP là kỹ thuật hướng các đối tượng ngoài thực tế vào trong ngôn ngữ lập trình. |
| So sánh Class và Object | |  |  | | --- | --- | | Class | Object | | Class là khuôn mẫu được định nghĩa để tạo ra các đối tượng | Object là thể hiện cụ thể của một class | | Dùng từ khóa class để khai báo lớp | Dùng từ khóa new để khởi tạo đối tượng | | Trong cùng package thì không được tạo hai class cùng tên | Đối tượng có thể khởi tạo nhiều lần từ 1 lớp | |
| Constructor trong Java và đặc điểm | Constructor là phương thức dùng để khởi tạo đối tượng.  Đặc điểm:   * Tên phải trùng với tên class * Không có kiểu dữ liệu trả về * Trong một class có thể có nhiều constructor * Mặc định không khai báo constructor nào thì Java sẽ 1 constructor mặc định không tham số, body rỗng * Có thể gọi constructor khác thông qua từ khác this(…), đặc biệt lưu ý là this(…) phải được gọi đầu tiên trong nội dung của phương thức khởi tạo |
| So sánh Constructor và Method | |  |  | | --- | --- | | Constructor | Method | | Không có kiểu dữ liệu trả về | Có hoặc không có kiểu dữ liệu trả về | | Trùng tên với class | Có thể trùng hoặc khác với tên class | | Nếu không khai báo thì Java sẽ cung cấp 1 constructor mặc định | Không có phương thức mặc định | | Chỉ được gọi 1 lần khi khởi tạo | Có thể gọi nhiều lần | |
| Getter và Setter | Getter: là phương thức trả về giá trị  Setter: là phương thức thiết lập lại giá trị |
| Từ khóa this | Được sử dụng để đại diện cho đối tượng hiện tại |

**5. Access modifier & Static**

|  |  |
| --- | --- |
| Access modifier? | Access modifier là các từ khoá được sử dụng để quy định phạm vi truy cập đến lớp và các thành phần của lớp |
| Các mức truy cập | • public: có thể truy cập từ bất cứ đâu  • protected: có thể truy cập trong cùng 1 package và các lớp con kế thừa  • default: có thể truy cập trong cùng 1 package  • private: có thể truy cập trong cùng 1 lớp |
| Static?  Cách truy cập  Mục đích sử dụng | Static: là từ khóa khai báo thuộc tính, phương thức, class. Khi đó thuộc tính và phương thức khi được khai báo với từ khóa static sẽ chịu sự quản lý trực tiếp từ class, không phải object.  Truy cập thành phần static thông qua tên class hoặc tên đối tượng của lớp  Mục đích:   * Định nghĩa thuộc tính, phương thức dùng chung cho đối tượng của 1 lớp * Tạo ra các lớp tiện ích |
| Các ràng buộc khi sử dụng static | Phương thức static chỉ có thể gọi được phương thức/ biến static khác trong một lớp  Phương thức bình thường có thể gọi được phương thức/ biến bình thường hoặc static  Dùng khối mã static{} để khởi tạo giá trị cho biến static |
| Các loại biến trong Java | Có 3 loại biến trong Java :   * Biến local : * Biến khai báo trong method, constructor và block * Không dùng được access modifier, phải khởi tạo giá trị mặc định mới dùng được * Chỉ tồn tại trong phạm vi được khai báo * Biến instance : * Biến được khai báo trong lớp và ngoài method, constructor, block * Dùng được access modifier, không cần khởi tạo giá trị ban đầu * Tồn tại trong suốt thời gian tồn tại của đối tượng * Biến static : * Khai báo bằng từ khóa static và khai báo trong 1 lớp * Được gán giá trị mặc định tương ứng với kiểu dữ liệu của chúng * Tồn tại trong suốt thời gian chạy chương trình |

**6. Inheritance**

|  |  |
| --- | --- |
| Tính kế thừa?  Đặc điểm | Tính kế thừa là cơ chế cho phép 1 lớp con sử dụng lại các đặc điểm hành vi đã được định nghĩa trước đó ở lớp cha.  Mục đích: tái sử dụng mã nguồn.  Đặc điểm:   * Lớp con kế thừa nhưng vẫn đảm bảo quy tắc access modifier * Không có kế thừa constructor * Java không hỗ trợ đa kế thừa * Mối quan hệ kế thừa là is-a |
| Từ khóa super và final | super: sử dụng ở lớp con dùng để gọi constructor, method, thuộc tính ở lớp cha.  Dùng final cho class để tạo lớp không có lớp con. |
| Lớp Object | Là lớp gốc của tất cả các lớp trong Java.  toString() là phương thức trả về 1 chuỗi biểu diễn thông tin của đối tượng. |
| Tính đa hình | Là 1 đối tượng có thể hiện/ hành vi theo nhiều cách khác nhau phụ thuộc vào ngữ cảnh.   |  |  | | --- | --- | | Overriding(Ghi đè) | Overloading(Nạp chồng) | | Overriding là quá trình định nghĩa một phương thức trong lớp con (subclass) có cùng tên, cùng kiểu trả về và cùng danh sách tham số với phương thức trong lớp cha (superclass). | Overloading là quá trình định nghĩa nhiều phương thức cùng tên trong một lớp, nhưng với số lượng hoặc kiểu tham số khác nhau. | | Phương thức Overriding phải cùng tên, cùng danh sách tham số, cùng kiểu trả về |  | | Access modifier ở phương thức con phải có level bằng hoặc cao hơn so với phương thức lớp cha |  | | Dùng final để cấm ghi đè phương thức ở lớp cha |  | | Xảy ra ở class có mối quan hệ kế thừa is-a | Xảy ra trong 1 class | | Đa hình tại run-time | Đa hình tại compile | |
| Ép kiểu | Ép kiểu là cơ chế chuyển đổi 1 tham chiếu đến đối tượng thuộc loại này thành tham chiếu đến đối tượng thuộc loại khác  Có 2 loại ép kiểu:  Ép kiểu ngầm định(Implicitcasting): ép từ subtype lên supertype  Ép kiểu tường minh(Explicitcasting): ép từ supertype xuống subtype |

**7. Abstract class & Interface**

|  |  |
| --- | --- |
| Tính trừu tượng  (Abstraction) | Tính trừu tượng là một khả năng ẩn các chi tiết trình triển khai và chỉ hiển thị tính năng tới người dùng.  Trong Java tính trừu tượng được thể hiện qua Abstract class và Interface. |
| Abstract class là gì?  Đặc điểm | Lớp trừu tượng (abstract class) trong Java là một lớp không thể được khởi tạo trực tiếp, chứa ít nhất một phương thức trừu tượng (abstract method) mà các lớp con phải triển khai.  Đặc điểm:   * Không thể tạo được đối tượng * Một lớp chứa method abstract thì lớp đó bắt buộc phải là lớp abstract (ngược lại thì không) * Lớp abstract có thể chứa các thuộc tính/method bình thường * Lớp abstract có thể extends/implement từ abstract khác/interface khác |
| Interface là gì?  Đặc điểm | Là 1 bản thiết kế của lớp định nghĩa các phương thức trừu tượng mà các lớp khác phải triển khai  Đặc điểm:   * Không thể tạo đối tượng từ interface * Các trường trong interface đều là hằng * Không có constructor * Method toàn bộ là abstract * Có thể extends từ 1 hoặc nhiều interface khác * Đại diện cho mối quan hệ has-a(can-do) |

**8. Clean Code ( Bài đọc thêm )**

|  |  |
| --- | --- |
| Clean Code là gì?  Đặc điểm | Clean Code (Mã sạch): là thuật ngữ để chỉ đến những mã nguồn “tốt”  Đặc điểm:   * Đơn giản * Trực tiếp * Dễ đọc * Dễ cải tiến * Có unit test và acceptance test * Các định danh đều thể hiện rõ nghĩa * Có ít sự phụ thuộc * Không có mã bị trùng lặp(duplicate) * Thể hiện được ý tưởng của thiết kế |
| Quy tắc của Clean Code | Định danh  Hàm  Format mã nguồn theo đúng Coding Convention  Xây dựng các Unit Test |
|  | Xem thêm ở bài này ở CodeGymX |

**9. MVC**

|  |  |
| --- | --- |
| MVC là gì?  Đặc điểm các thành phần | MVC là một mẫu thiết kế phần mềm phổ biến được sử dụng để tách biệt dữ liệu (Model), giao diện người dùng (View) và logic điều khiển (Controller).  Đặc điểm:   * Model chứa các đối tượng dữ liệu và logic kinh doanh. Nó không biết gì về View hay Controller. * View hiển thị dữ liệu model cho người dùng. Nó chỉ biết về Model nhưng không biết Controller. * Controller xử lý các request từ người dùng, tương tác với Model để lấy dữ liệu và truyền dữ liệu đến View để hiển thị. |

**10. DSA: List**

|  |  |
| --- | --- |
| Java Collection famework | Collections trong Java là 1 khuôn khổ, cung cấp 1 kiến trúc dùng để lưu trữ và thao tác tới nhóm các đối tượng.  Các hoạt động dựa trên 1 tập hợp CRUD, tìm kiếm, sắp xếp  Java Collection cung cấp nhiều interface :   * List: cấu trúc dữ liệu tuyến tính, trong đó các phần tử được sắp xếp theo 1 thứ tự nhất định, cho phép các phần tử trùng lặp. * Set: lưu trữ không có phần tử nào trùng nhau và tập hợp chưa được sắp xếp. * Queue: hàng đợi hoạt động theo cơ chế vào trước ra trước (FIFO) * Stack: ngăn xếp hoạt động theo cơ chế vào trước ra sau (FILO) |
| ArrayList | Phân biệt Array và ArrayList :   |  |  | | --- | --- | | Array | ArrayList | | Kích thước cố định | Kích thước thay đổi được | | Chỉ có phương thức length | Có nhiều phương thức hỗ trợ | | Có thể lưu trữ kiểu dữ liệu nguyên thủy và đối tượng | Chỉ lưu trữ được kiểu đối tượng (kiểu nguyên thủy tự động chuyển sang đối tượng) | | Tốc độ lưu trữ thao tác sẽ nhanh | Chậm hơn | |
| LinkedList | Phân biệt ArrayList và LinkedList:   |  |  | | --- | --- | | ArrayList | LinkedList | | Sử dụng các mảng động để lưu trữ các phần tử | Sử dụng danh sách liên kết đôi để lưu trữ các phần tử | | Truy xuất ngẫu nhiên sẽ nhanh hơn | Thêm/xóa đầu nhanh | |
| Các phương thức làm việc | add(): thêm phần tử  get(): truy xuất phần tử  remove(): xóa phần tử  size(): kích thước của danh sách |

**11. DSA: Stack & Queue**

|  |  |
| --- | --- |
| Generic là gì?  Ưu và nhược điểm. | Generic là cơ chế cho phép sử dụng kiểu dữ liệu như là tham số (tham số hóa kiểu dữ liệu.  Ưu điểm:   * Phát hiện lỗi ngay tại thời điểm biên dịch * Không cần ép kiểu dữ liệu * Xây dựng các thuật toán tổng quát, tái sử dụng   Nhược điểm:   * Không thể khởi tạo đối tượng * Không thể sử dụng static * Không thể sử dụng kiểu dữ liệu nguyên thủy * Không thể sử dụng exception |
| Stack là gì?  Các phương thức làm việc với Stack | Stack: là cấu trúc dữ liệu dạng danh sách, thêm và lấy phần tử theo quy tắc FILO.  Các phương thức:   * push(): thêm * peek(): lấy ra xem * pop(): lấy ra xem và xóa * empty(): kiểm tra rỗng * search(): tìm kiếm |
| Queue là gì?  Các phương thức làm việc với Queue | Queue: hàng đợi là cấu trúc dữ liệu danh sách, thêm và lấy phần tử theo quy tắc FIFO.  Có 3 lớp triển khai Queue: LinkedList ; PriorityQueue ; ArrayDeque  Các phương thức:   * **offer()** , add() : thêm * **peek()** , element() : lấy ra xem * **poll()** , remove() : xóa * Các phương thức bôi đen sẽ có trả về, còn phương thức không bôi đen sẽ ném ngoại lệ |

**12. Java Collection Framework: Map & Tree**

|  |  |
| --- | --- |
| Map là gì?  Đặc điểm | Map sử dụng để lưu trữ và truy vấn dữ liệu theo cặp khóa key-value  Đặc điểm:   * Mỗi cặp key-value được gọi là entry * Map không cho phép 2 key trùng nhau * Mỗi key tương ứng với 1 value |
| HashMap | Giống với đặc điểm của Map ở trên  Điểm khác:   * Không đảm bảo thứ tự entry thêm vào * Cho phép 1 key là null * Cho phép nhiều value có thể null |
| LinkedHashMap | Giống với HashMap  Điểm khác: Duy trì phần tử entry theo thứ tự thêm vào |
| TreeMap | Giống với Map  Điểm khác:   * Không có key giá trị null * Duy trì các phần tử được thêm vào theo thứ tự key được sắp xếp (mặc định sắp xếp tăng dần)   Lưu ý: với key là kiểu Object do người dùng tự định nghĩa thì cần phải triển khai interface Comparable hoặc Comparator |

**13. Search Algorithm**

|  |  |
| --- | --- |
| Linear Search | Thuật toán tìm kiếm tuần tự (Sequential Search) là một phương pháp tìm kiếm một phần tử cụ thể trong một tập hợp dữ liệu, chẳng hạn như một mảng hoặc danh sách liên kết. Thuật toán thích hợp cho tập hợp chưa được sắp xếp và số lượng phần tử ít.  Ý tưởng:   1. Bắt đầu từ phần tử đầu tiên của tập hợp dữ liệu. 2. Kiểm tra phần tử hiện tại có bằng với phần tử cần tìm hay không. 3. Nếu tìm thấy, trả về chỉ số của phần tử đó. 4. Nếu không tìm thấy, di chuyển đến phần tử tiếp theo và lặp lại quá trình. 5. Kết thúc nếu kiểm tra qua tất cả các phần tử và không tìm thấy, trả về -1.   Độ phức tạp:   * Tối ưu: 1 * Tệ nhất: n |
| Binary Search | Tìm kiếm nhị phân (Binary Search) là một thuật toán hiệu quả để tìm kiếm một phần tử trong một tập hợp dữ liệu đã được sắp xếp (thường là một mảng hoặc danh sách).  Cây nhị phân (Binary Tree) là một cấu trúc dữ liệu trong lập trình, bao gồm một tập hợp các nút (nodes) liên kết với nhau theo cách mà mỗi nút có tối đa hai con, một con ở bên trái và một con ở bên phải. Nút bên trái sẽ luôn nhỏ hơn nút bên phải.  Ý tưởng:   1. Bắt đầu với tập hợp dữ liệu đã sắp xếp. 2. Chọn phần tử giữa của tập hợp (phần tử ở vị trí mid). 3. So sánh phần tử giữa với phần tử cần tìm. 4. Nếu phần tử giữa trùng với phần tử cần tìm, trả về chỉ số của phần tử đó. 5. Nếu phần tử giữa lớn hơn phần tử cần tìm, tìm kiếm trong nửa bên trái của phần tử giữa. 6. Nếu phần tử giữa nhỏ hơn phần tử cần tìm, tìm kiếm trong nửa bên phải của phần tử giữa. 7. Lặp lại quá trình cho đến khi tìm thấy phần tử hoặc không còn phần tử nào để tìm. 8. Kết thúc nếu không tìm thấy, trả về -1.   Độ phức tạp:   * Tối ưu: 1 * Tệ nhất: log n |

**14. Sort Algorythm**

|  |  |
| --- | --- |
| Bubble Sort | Ý tưởng: Duyệt qua danh sách từ đầu đến cuối và so sánh lần lượt các cặp phần tử liền kề, đổi chỗ chúng nếu chúng không được sắp xếp theo thứ tự mong muốn. Lặp lại quá trình này cho đến khi không còn phần tử nào cần đổi chỗ.  Độ phức tạp:   * Tối ưu: n * Tệ nhất: n^2 |
| Selection Sort | Ý tưởng: Chọn phần tử nhỏ nhất từ danh sách chưa sắp xếp và đặt nó vào đầu danh sách đã sắp xếp. Lặp lại cho đến khi danh sách đã sắp xếp hoàn toàn.  Độ phức tạp:   * Tối ưu: n^2 * Tệ nhất: n^2 |
| Insertion Sort | Ý tưởng: Chia mảng thành hai phần: phần đã sắp xếp và phần chưa sắp xếp. Duyệt qua phần chưa sắp xếp, lấy từng phần tử và chèn nó vào phần đã sắp xếp ở đúng vị trí.  Độ phức tạp:   * Tối ưu: n * Tệ nhất: n^2 |

**15. Xử lý ngoại lệ & Debug**

|  |  |
| --- | --- |
| Exception là gì? | Exception (Ngoại lệ) : là sự kiện bất thường xảy ra trong quá trình thực thi chương trình Java, nó phá vỡ flow(luồng xử lý) bình thường của chương trình, thậm chí gây chết chương trình. |
| Phân loại Exception. | Có hai loại Exception:   * Checked Exception: xảy ra lúc compile time, không thể bỏ qua lúc compile, bắt buộc lập trình viên phải xử lý * Ví dụ: IOException, FileNotFoundException,.. * Unchecked Exception: xảy ra tại thời điểm thực thi chương trình, lỗi không chắc xảy ra, lập trình viên có thể bỏ qua. * Ví dụ: NullPointerException, ArithmeticException.   Error: là lỗi nghiêm trọng liên quan đến chương trình thực thi (JVM) của ứng dụng hệ thống, có thể gây chết chương trình. |
| Các cách xử lý ngoại lệ | Try-catch / finaly: xử lý ngoại lệ ngay tại đoạn mã bị lỗi  Dùng throw/ throws để ném ngoại lệ cho method/logic khác xử lý   * Lưu ý khi dùng try-catch / finaly: * Khối có thể không có khối catch hoặc nhiều catch * Tại một thời điểm chỉ xảy ra một ngoại lệ và chỉ có 1 khối catch được thực thi * Khối catch phải được sắp xếp từ cụ thể đến chung nhất * Khối finaly luôn luôn được thực thi dù cho chương trình có xảy ra ngoại lệ hay không, khối finaly dùng để giải phóng tài nguyên |

**16. IO: Text File**

|  |  |
| --- | --- |
| Khái niệm stream và phân loại stream | Các hoạt động nhập/xuất dữ liệu gọi là stream.  Phân loại: Character stream: được thiết kế để nhập/xuất dữ liệu theo ký tự Unicode  Byte stream: hỗ trợ nhập xuất theo dạng byte(đọc ghi nhị phân. |
| Các lớp triển khai hay dùng của Character stream | |  |  |  | | --- | --- | --- | | Abstract class | Class triển khai | Class tạo bộ nhớ đệm | | Reader | FileReader | Buffered Reader | | Write | FileWrite | Buffered Write | |
| Giới thiệu về CSV và cách đọc ghi CSV file | CSV là viết tắt của "Comma-Separated Values" (Giá trị được phân tách bằng dấu phẩy). |

**17. IO: Binary & Serialization**

|  |  |
| --- | --- |
| Cơ chế Serialization | Là cơ chế tuần tự hóa, cho phép chuyển trạng thái của một Object về một byte stream  Ngược lại chuyển từ byte stream về một Object được gọi là de-serialization |
| Những class triển khai đọc ghi byte stream | |  |  |  |  | | --- | --- | --- | --- | | Abstract class | Class hay dùng | Class đọc ghi | Bộ nhớ đệm | | InputStream | FileInputStream | ObjectInputStream | BufferedInputStream | | OutputStream | FileOutputStream | ObjectOutputStream | BufferedOutputStream | |
| Một số lưu ý khi triển khai | Thuộc tính áp dụng từ khóa static thì sẽ không đạt được Serialization  Class implement interface Serializable thì các lớp con không cần implement nữa  Thuộc tính của đối tượng phải implement Serializable |

**19. String & Regex**

|  |  |
| --- | --- |
| String và đặc điểm của String | String dùng để lưu trữ làm việc với chuỗi văn bản trong Java  Khai báo biến String có hai cách:   * Gán trực tiếp thông qua tán tử gán “=” (giống với kiểu nguyên thủy) * Thông qua từ khóa new (giống với Object) * Khuyên dùng cách 1   Một số phương thức String: length(), concat(), subString(), split(), charAt(), trim() ,… |
| StringBuider và StringBuffer | Cả hai đều thay đổi được mutable  Điểm khác nhau:   * StringBuider không hỗ trợ đồng bộ * StringBuffer có hỗ trợ đồng bộ trong ứng dụng đa luồng   Phương thức: append(), toString(), reverse(), delete() ,.. |
| Regular Expression | Biểu thức chính quy là một chuỗi được sử dụng để quy định dạng thức của một chuỗi |