Lecture 1

Computer System < Hardware e.g. CPU. disks / Software < System ~ (control hardware) / Application ~ (for end user)

software. Application. system software. users.

Hardware. CPU RAM Disk

Computer System & program
Java and JVM
Soft design Principle
OOP 面向对象

Program: 一串指令进行计算 Fetch → Decode → Execute → Fetch → ...
→ Fetch - Decode - Execute Cycle. 冯诺依曼 John Von Neumann.

① Machine-language instruction is hard for human.
② Low level language provides a level of abstraction on top of machine code. * e.g. Assembly (汇编) language.
③ High level language provides stronger abstraction and resembles more of natural language * e.g. C++ Java Python etc. (在Java JDK中)

Java program → Java Compiler → Java Bytecode → Java Virtual Machine → low-level machine code.

.java — JVM — .class — JVM — e.g. Win. Mac

.class → Class Loader Subsystem → load .class / .jar (Jar is sets of .class)
Runtime Data Area
output ← Execution Engine.

[Jar: Java ARchive. groups of .class.]
.jar → standard java packages. e.g. Java.io.
.class and .jar → classpath 中的 dependency.
ERROR occur when class loader can't locate a required .class.

Store all kinds of data and information.
(class level 常量和静态变量) → Method Area / Objects and instance → Heap Area
local variable → Stack Area).

JRE: Java Runtime Environment
JDK: Java Development kit
JDK (JRE (Core class, JVM), Compiler, Debugger) → development kit.

① Translate "run anywhere" .class to "run on this particular machine" instructions
② by Interpreter and JIT Compiler (also for optimization)
③ garbage collector identifies objects that are no longer in use and reclaim Memory.

Soft Design Principles: High Cohesion 高内聚. Low Coupling 低耦合. Information Hiding 信息隐藏

Modules: pieces of Complex software system
Cohesion: elements of a module functional related to each other
Coupling: different modules depends on each other

→ easy to understanding, extend and modify
otherwise → difficult to read, understanding reuse. test. and maintain.

users. → Interface. → Core Info.
Increasing coupling → breaking information hiding.

过程化思维 → 面向对象思维 / 出问题后 抽象组合
将 function / data 封装在 class 中, 并使用 Access Control mechanism in Java

OOP { Encapsulation (封装), Inheritance (继承), Abstraction (抽象), Polymorphism (多态)

Visibility
- At the top level (class or interfaces)
  - package-private (default): visible only within its own package
  - public: visible to all classes everywhere
- At the member level (fields or methods)
  - private: can only be accessed in its own class
  - package-private (default): visible only within its own package
  - protected: can be accessed within its own package and by a subclass of its class in another package.
  - public: visible to all classes everywhere

Rule of thumb: 最严格使用.

Getter and Setter:
getxxx() → read
Set xxx() → modify.

The Object class is the parent class of all the classes.

@ override
```
public boolean equals (Object o) {
    Money other = (Money) o;
    return this.amount == other.amount; }
}
```

@ Override
```
public String toString () {
    return "Amount is" + amount;
```

Abstract class < ① 不可实例化 ② 可以有抽象方法 / 非抽象方法
Interface → group of related abstract method with empty body.

a class can implement multiple interfaces but can inherit only one abstract class.

Polymorphism < An object can take many form. / Some operation performed in different way.
shape < Rectangle, cycle. .draw() — add(1,2) / add(1,2,3)

Binding < Static Binding (early) compile time e.g. Method overloading / Dynamic Binding (late) execution time. e.g. Method overriding
name of method → final implementation.

Example | Term
List<E> | Generic type. 泛型
E | formal type parameter / Type varible.
List<String> | Parameterized type
String | actual type parameter
List | Raw Type.

```
List list = new ArrayList();
```
⚠ warning not error

```
public class Pair<T> {
private T first; private T second;
public Pair() { first = null; second = null; }
public Pair(T first, T second) {this.first = first; this.second = second; }
public T getFirst() { return first; }
public T getSecond() { return second; }
void setFirst(T first) { this.first = first; }
void setSecond(T second) { this.second = second; }
}
```

classes in Java collections (e.g. List / Queue / Set) are typically generic classes.

<T extends Bounding Type>
<T extends Animal & Comparable> } 限定T类型 - 类是要实现 - 接口 - 类 - 如何是接口.
a class must be the first one.
Bounding Type can be a class and a interface.

List<String> 和 List<Object> 没有关系, 因此使用
List<?> 来替换! ? 是通配符.
List<?> 是所有 List<T> 的父类
List<? extends T> 是所有 T 的子类
List<? super T> 是所有 T 的父类

为了兼容性 (Compatible) Java 的泛型采用 伪泛型策略 (the strategy of pseudo generics)
→ type erase 类型擦除 compilation 时检查, 在JVM中无泛型
e.g. <T> → Object <T extends Number> → Number

ADT: Abstract Data Type
定义: 数据类型, 它们为定义为数据和操作
数据 data 和操作 operation 都封装于 client 后.
e.g. List / Stack / Queue / Set / Map
Operations: Creator, Producer, Observer, Mutators
build new; build new from the old one; different type return; change the object.

✓ ArrayList. LinkedList → 双向链表. equals() 重写 → hashCode() 重写
HashSet 不 order. 最高 performance 最多-1 null
LinkedHashset insertion order 中间 最多-1 null
Tree Set natural sorting order. 最慢 doesn't allow even single element.
TreeSet / Map ⊗ null
All are Serializable. - All have fail-fast iterator
None are synchronized. → no safe.)

Collections class 包含很多算法 - 些静态方法 (Static Method).
ClassCastException → not Comparable
sort (list) / binarySearch (list key) / min (collection) / max / fill (list E) / copy (dest, src) / reverse (list) / shuffle (list)

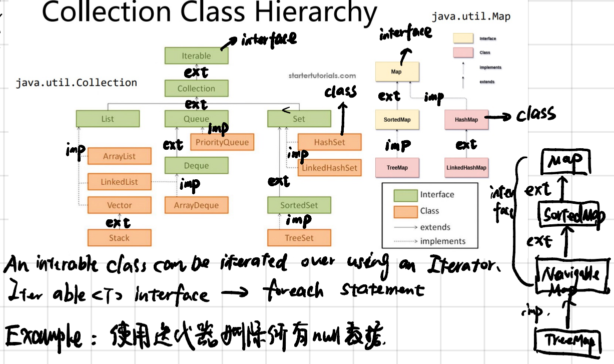
public class xxx Comparator implements Comparator<T> {
public int compare (T o1 T o2) ...
}
Collections. sort (xxx, new xxxComparator())

Arrays. asList (T... a) 把 T...a 变成 List.
e.g. List<String> list = Arrays.asList (new String [size]);
Collections.nCopies (int n, T o) immutable list of n copies of the object o. List.addAll...
→ removeAll (Collections).
Collections.singleton (T o) immutable set
Collections.empty xxx Set / List / Map

Lecture 2

Generics
Abstract Data Type (ADT)
Collections.

Collections 是一个类 Collection 是一个接口.



An iterable class can be iterated over using an Iterator.
Iterable<T> interface → foreach statement

Example: 使用迭代器删除所有 null 数据.
```
for ( Iterator<?> i = c.iterator() ; i.hasNext(); ) {
    if (i.next() == null) {
        i.remove();
    }
}
```

2个 sets 相等 当且仅当 Size 相等, 元素相同. set 的 hashcode 定义为所有内部元素 hashcode 的和.

Generics (泛型) 没有泛型的缺陷 < ① Explicit casting is required → inefficient & hard read. ② error-prone (易错的), may cause type-related runtime errors if programmer make a mistake when the explicit casting.
→ Introduce in JDK 5.0

public class ArrayList<E> | E (sometime. T) any non-primitive type
public boolean add (E e) | All element in the List is of type E.
public E get (int index)

principle: discover errors as early as possible.

e.g. ArrayList<String> list = new ArrayList<String>();
list.add ("Hello!"); ✓ list.add (2022); ✗. ← no explicit cast here.
String tmp = list.get(0); ✓ ← complier will add the correct type cast.

Set: containsAll() / addAll() / retainAll() / removeAll()
boolean / set (∨) / set (∧) / set (-)
new Hashset<>(c) c is Collection<Type>. 类型
Collection <Type>

List: indexOf (object o) / lastIndexOf (Object o)
subList (from, to) / addAll ()

{ entrySet() / keySet() }
Map: Map<key, Val> : for (interator<key> i = M.keySet().interator() ...)
entrySet() 是 key 和 val 可用 getKey 和 getVal 获得.

Functional programming is a programming paradigm (编程范例)

面向对象 面向过程 函数

OOP. Procedural P.. Functional P...

① functions are treated like any other Varibles { be passed / assigned / returned.

② Higher order function take another function as argument, return another function as result. → data passing.

③ { No Side Effect (pure can't change external states) / pure always produce same output for the same input (regardless of the history) → variable 一经定义其值不变.

④ pure don't have assignment statement. Immutability

⑤ no while or for loop, → using recursive / higher order funct.

① easy debug, test, parallelize. { Same in - Same out (确定性) / No side effects / Data immutable.

② architecture simplicity

使得使用API时可以不关注外部状态.

Lambda Expressions (Java 8)

(函数式接口的简化) → 只有一个抽象方法的接口

(param1, param2) → { lambda exp }

1. 匿名函数
2. 不属于任一类
3. 可作为参数传给另一个方法
4. 可以在需要时执行.

· () is needed if no parameter

· 括号可省当且仅有一个 parameter时.

i / return / if / for 用; 与多语句

Single expression → 可省 {}

Functional Programing / Lambda Expression

e.g. Collections.sort (strList, (s1, s2) → Integer.compare (s1.length(), s2.length()));

Lambda 与 Comparator.compare (T o1, T o2) 匹配 / Comparator<? super T> 匹配.

参数匹配由编译器完成, 但不会从 raw type 推断. 可推断的可省略.

Lambda 和 抽象 interface 匹配, 要求只有一个抽象方法. (可有多个 default / static 方法)

The local variable could be accessed without changing it value.

Method { static Method → TypeName :: staticMethod. / Instance Methode. { bound: Instance Name :: Method Name. / unbound: className :: Method Name. } / Constructor → className :: new(); 后参数使用第一个参数的

① 创建出函数接口"实例" MyInterface ref = Lbdam ref. ...

② 在访问元素中进行同一操作 strList.forEach (elem → system.println(...));

重要函数式接口 Java. ① Consumer<T> → void accepted (T t);

System.out :: println ② Supplier<T> → T get();

Instant :: now ③ Predicate<T> → boolean test (T t);

Collection :: isEmpty ④ UnaryOperator<T> → T apply (T t);

String :: toLowerCase ⑤ Binary Operator<T> → T apply (T t1, T t2);

BigInteger :: add ⑥ Function <T, R> → R apply (T t);

Array :: asList

Serializable Interface (also called marker / tagging interface)

transient keyword (e.g. ~ int age)

Path create: Path P1 = Paths.get ("___")

Stream API (from Java 8) source → intermediate operation1 → intermediate operation2 → ... → Terminal operation

stream stream stream

create a stream { xxx.stream(); / Stream.generate(xxx) → Supplier<T> 接口. / function → 生成无限流. / Stream.of (T... values) / int[] → List<Integer> IntStream.of (arr).boxed().toList(); → 相当值流 (primitive Type Stream) Long、Double. 不使用包装类.

Lecture 4

stream API / Optional <T>

Intermediate Operations → Lazy: 所有流操作不会执行直到终止操作执行.

filter() Predicate. boolean test()
map() Function R apply (T t) / flatMap. Function
distinct() sorted() → natural order <? super T, optional<U>>.)
sorted() Comparator <? super T> comparator)

Terminal Operation → 终止流, 可以获取结果

anyMatch() allMatch() noneMatch Predicate → boolean

findFirst() → return Optional <T>.

collect() { 收集到集合 Collections.toSet / toList / toCollection (xxx::new) HashSet / 收集到数组 Stream.toArray (xxx[]::new)

Optional <T> 避免 null pointer exception

Reduction { min() max() average() sum() / reduce() ← general one BinaryOperator <T>

Create Optional Values { Optional.of (T value) 当 value 是 null时会报错. / Optional.empty() / optional.ofNullable (T value)

Java I/O and File are in Java.io Package.

I/O { Input / Output } { Byte Stream { InputStream 读取 bytes序列 / Output Stream 写到 bytes序列 } / Character Stream → Internationalize. }

其中 Reader / Writer 抽象方法 / Sub class xxxReader / xxx Writer. read() write()

Internationalization (I18N) 使程序有不同语言的支持.

< Reader / Writer 使用 char 描述 class. base on char 而不是 byte.

Every Platform Environment must support basic Standard charset.

ASCII → 7bit 128 / 8bit 256

GB2312 → 2byte GBK GB18030 extend extend.

Unicode { UTF-8 Unicode Transformation Format 8 bit / UTF-16 Java char Unicode code point / UTF-32 } 标准

int ↔ char 使用 unicode mapping

(U+0000 U+10FFFF) > U+FFFF called supplementary characters

greater than U+FFFF a pair of char value 16+16 = 32 bit = 4 byte.

Java System / platform default encoding → default encoding when JVM starts. → Differ. OS and language settings changed. 可

File encoding → 特定 Java → 可以改变 → 使用Java读文件时. Java system default encoding 和 文件 decoding 应该一致. when inconsistent → charset. forName ("___") 与编码一致.

Lecture 5

FilterInput Stream (an example of the Decorator design pattern.)

1. 包含其它 InputStream 作为 basic source of data.
2. InputStream 的子类提供若干功能.

e.g. gzip + buffered origin data

... = new GZIPInputStream (new BufferedInputStream (new FileInputStream ("xxx")))

Text I/O { Scanner / PrintWriter } ② new Scanner ("xxx");

using Scanner for reading

File file = new File ("xxx");

Scanner in = new Scanner (file);

while (in.hasNextDouble()) {
double value = in.nextDouble();
}

using PrintWriter for writing text files

printwriter out = new Printwriter (xxx);

out.println ("Hello World!");

out.printf (" xxx : %8.2f", total);

I/O Command Line. { standard Stream e.g. system.out / Console e.g. System.console }

{ System.err / System.in → byte stream / System.out }

BufferedReader br = new BufferedReader (new InputStreamReader (System.in))

str = "" while (!str.equals("quit")) {
str = br.readLine(); System.out (str);

3. System.out PrintStream byte stream 但实现了 character 的特征.

→ print / println / format ...

使用 setOut() 重定向: PrintStream out = new PrintStream (new File (" ")) system.setOut (out)

System.out.println.

→ 所有东西在控制台输出打印. System.out.println ("..."); → 难以识别打印. → 使用日志 logging → java.util.logging

I/O isn/char error Encoding Coupling R/W Command Line. InputStream

Lecture 6

persistence serialization Files Exception

· 当前目录 .. 上一级目录.

从 / 开头 → 相对. normalize() 去除冗余.

以 C:/ 等 root开头 → 绝对.

toString() / getFileName() / getName(0) / getNameCount()

subpath(0, 2) / getParent() / getRoot()

toAbsolutePath() → 不一定存在, 绝对路径, 不检查

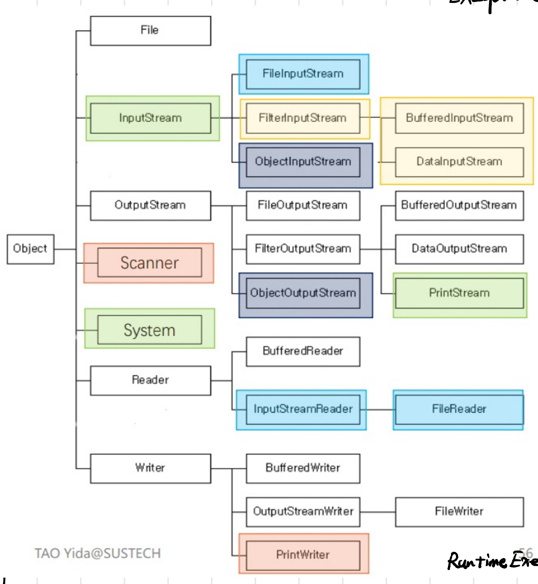
toRealPath() → 一定存在, 绝对路径, 检查.

Files.list (Path path) → 所以不进入子目录的 Path (子目录 / 文件)

Files.walk (Path path) → 深度优先 列出所有子目录 (子目录 / 文件)

object → Throwable. → 才可以 throw / catch.

Error / Exception



RuntimeException and its subclass: are unchecked exceptions. Others are checked exceptions.

{ checked Exception → 编译时检测 / unchecked Exception → 可忽略编译检查 }

InputStream in = new FileInputStream () ...

Reader reader = new InputStreamReader (in, "UTF-8");

OutputStream 与 Writer 同理.

try-with-resources 自动关闭.

try (Scanner scanner = new Scanner (...)) {
...
} catch (File ...) {
}

@ override close() AutoCloseable.

Process 进程: Executing a program starts a process why: 单继承

Thread 线程: A process can have multiple threads. Threads within a process share the memory and resources of the process.

A extends B. Hunt A extend Thread?

main 线程 → 自动开启, 可以创建其它线程 (extend Thread ✗ / implement Runnable ✓)

override run() method. → 本质是在 Runnable 中的 run.

→ 我们在 run 中实现方法. 而. start() 才会真正启动线程

why start() not run()? → Start() is non-blocking, Don't have to wait for it before executing the subsequent operations. 普通方法, 只是使用 run(). Thread thread = new Thread(Dog)

Thread 有 stand 方法, implements Runnable 的没有 start 方法.

Runnable run = () → {…} Thread dog = new Thread(run)

Ext [Thread] imp — Thread class / Runnable interface. override run() method

Thread Safety. → synchronization

→ 有多个线程访问 (同一时间段访问同个地址资源) 在访问, 其他线程会被访问)

不允许访问. [互斥锁] → 对象有锁标记 保证任一时刻只有一个访问

synchronized (对象) { … 同步代码 } 对代码块加锁.

public synchronized void method(…) { … }

在静态方法, 加在类本身上. [xxx.class] 多个线程获得共同对象 ↑ 当前类

定义一个 Lock: Lock lock = new ReentrantLock();

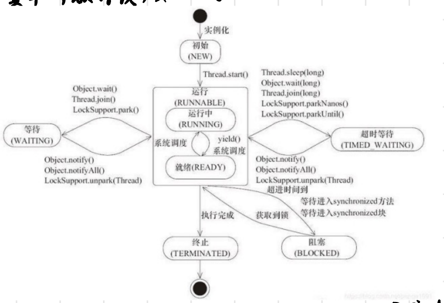
① 锁住当前 Lock: lock.lock()   try { … }

② 解锁当前 Lock: lock.unlock()   finally { lock.unlock(); }

Condition xxx Condition; xxx Condition = lock.newCondition()

xxx Condition.signalAll()   xxx Condition.await(); → 让当前程序 wait, 其他可以访问.

其中一个获得锁, 其他没有



所有 Collection classes 都不是线程安全的

{ fail-fast iterators } 3类线程安全

Copy On Write Collection
Compare and Swap Collection
Collections using Lock

CopyOnWriteArrayList is fail-safe interators

Concurrent xxx. Blocking Queue.

```java
public class MyBlockingQueue<E> {
    private Queue<E> queue = new LinkedList<>();
    Lock lock = new ReentrantLock();
    Condition notEmpty = lock.newCondition();
    Condition notFull = lock.newCondition();
    private int size;

    public MyBlockingQueue(int size) {
        this.size = size;
    }

    public void put(E element) throws InterruptedException {
        try {
            lock.lock();
            while (queue.size() >= size) {
                notFull.await();
            }
            queue.add(element);
            notEmpty.signal();
        } finally {
            lock.unlock();
        }
    }

    public E take() throws InterruptedException {
        try {
            lock.lock();
            while (queue.size() == 0) {
                notEmpty.await();
            }
            E output = queue.remove();
            notFull.signal();
            return output;
        } finally {
            lock.unlock();
        }
    }
}
```

Lecture 7 多线程

multithreading Overview

creating / starting Threads

Thread Safety

Concurrent Collections

IP Address: distinguish a device on int or local network.

Domain Name: human-friendly of IP. a.k.a DNS.

Port Number: identify different application / processes.

Network Protocols rules. format. transmit. receive. ⇒ Communication

HTTP (Hyper Text Transfer Protocal)
TCP (Transmission Control)
UDP (User Datagram Protocol)

Endpoint = IP + Port.

Socket (host, port) get outputStream() app. get inputStream Server

Server Socket (post) . accept(). get inputStream get outputStream

21. 25. 80 (8080). 110. 143.
443. 666. 989. 23. 25565. 27015 ⇒ Well known.

抓取网页 GET + resourse + HTTP/1.1 + Host: xxxx

try (Socket s = …) {
  Input Stream instream = s.getI…()
  Scanner In = new … (instream);
  printwriter out = new … (out…);
  String cmd = … out.print(cmd) out.flush().
  while in.hasNextLine() { String input = in.nextLine() print () … }

[URL] URL URLConnection HttpURLConnection

URL u = new …(url); URLCon… httpConn = (…) conn

int code = httpConn.getResponseCode()

String msg = httpConn.getMessage();

if (code != HUC. HTTP.ok) { … }

InputStream istream = httpConn.getI…S…

Scanner in = new Scanner(istream).

HTML: HyperTextMarkUpLanguage.

API interface for programs to Communicate

RESTAPI API conforms to the constraints of REST architecture style.

REST. → REpresentational State Transfer.
→ Software architecture Style.

GET POST PUT/PATCH DELETE.

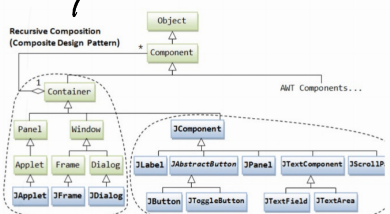
Lecture 8. 网络

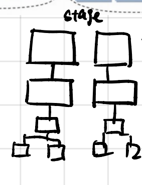
Network protocols
Socket program
Get Web Data.

GUI: Graphical User Interface.

→ AWT: Abstract Windows Tookit. → Button. Label. Text Feild (component) / Frame. Panel (Container)

Swing. JavaFX.





→ 窗体. (一个)

Scene 场景. 一个窗体同时可有多个 Scene. 只能切一个 stage.

Scene graphs 可视化组件 Tree. branch node / leaf node.

A JavaFx application must specify the root node for the scene graph by setting the root property.

JavaFx UI must access & modify from JavaFx Application Thread.

JavaFx.

```
public class … extends Application {
  @override
  public void start(Stage primaryStage) … {
    … primarystage.show();
  }
  public static void main(String[] args) {
    Application.lauch(args);
  }
}
```

JavaFx FXML
1. XML-based language
2. build interface separate from the application logic.
(EXtensible Markup Language)

Design code are mixed with the application code.
Code will be easier to maintain
if application design is seperated from the application logic

Lecture 9 Java Fx

Reflection 在 runTime 时检查或更改方法.类. 获取行为 actual type 对象.

e.g. x instance of ClassName | Class c = x.getClass()

class 对象是在 JVM 中被自动加载 (load) 不能手动创建. 且每个类只会创建一个 class 对象

Getting Class Fields (获得成员变量) class<> Accessible Constructor Field Method object

Field getField(name) → public member variable

Field getDeclaredField(name) → any …

Field[] getFields() / Field[] getDeclaredFields()

Field f = …. | f.getName() → 返回名字

f.getModifiers() | f.getType() → 返回类型

Modifier.isFinal / isprivate f.setAccessible(true) 使修改

f.set(bc, value)

Getting class Method

Method getMethod(name, Class…) [Class] → public method

Method getDeclaredMethod(name, class…) → any.

Method[] getMethods() Method[] getDeclaredMethods()

用法 class.getMethod("Substring", int.class); ① Risky ② compile time type safety ③ bugs refactoring ④ slower.

[Class] ① [Class] ② Object

Get Object Instance class.newInstance();

② constructor c = class.getConstructor(class1, class2)

className i = (Class Name) c.newInstance(val1, val2)

Getting class Inheritance → class.getSuperclass() class[] → class.getInterfaces()

JUnit / Spring framework (DI)

Annotation → metadata (data about data) ≠ Comments

Compiler / Build-time / Runtime instruction. catch err or warnings generate code. examine by Reflection

Build-in ~ → Java. Meta-an ~ → annotation

@Deprecated: 不再使用.废止.

@Override: 重载, 避免了方法签名是否与父类有同

@SuppressWarnings: 屏蔽无用警告. uncheck…

@SafeVarargs: 可变参数. 代码安全. 确认.

@Target 注释范围, 标注可用范围 / Java element

@Retention how… stored (at which level it is available) ⇒ 生命周期

@Repeatable 可以使用多注释在一个地方.

@inherited 注释类可以从父类继承

@Documented 在 javadoc 中文档化注释

Lecture 10 Reflection Annotation

[JavaEE [JavaSE [JavaME]]]

localhost:80 → localhost

Java EE Client: a web client / application client JavaServer Faces

WebTier: 客户端和业务 Servlet. JSP. JSF (Java Server Page)

solve or meet the need of a particular business domain

Business ~: 确定业务 handle interaction between client and business

POJO (plain old java object)
EJB (Enterprise Java Bean)

Data Tier: 存储 JDBC (~ Database Connectivity API)

JavaEE Server provides JPA (~ Persistence API)
services to these components JTA (~ Transaction API)
in the form of a container.

Application client Container / Web Browser — Client Machine

Servlet JSP page — Web Container
Enterprise Bean. EJB Container

Java EE Server Database.

class ← Servlet ↑ 线程安全 Generic Servlet ↑ HttpServlet ↑ My Servlet. doGet doPost doPut doDelete

Servlet Container ← 已请求 → 网络存储 创建时 Create

a → a. Hibernate

RI (Reference implementation) (framework)

ORM (object-Relational-Mapping)

Lecture 11 JavaEE

IoC (Inversion of Control)

DI (Dependency Injection) detection

@Component → automatic bean
→ ① Scan our application for classes annotated with @
② Instantiate them and inject any specified dependencies into them
③ Inject them whenever needed.

@Autowired → applied on setter / constructor.
injects object dependency implicitly
→ ① Spring container to automatically resolve dependencies between collaborating beans by inspecting the beans that have been configured.

@ Configurations → config by itself / define instance beans
@ Component Scan 确定扫描 create and return.
@ Bean bean: object instantiated, assembled, managed by Spring IoC Container. with @Configuration

Geting Class Object

Class cls1 = String.class ← 已知类名

class cls2 = Class.forName("…") ← 已知全名

Class cls3 = x.getClass() ← 已知类 对象实例

使用 cls1.getName() 获取名字

AOP (Aspect - Oriented - Programing) : 面向切片编程，通过预编译方式和运行期…
MVC (model - view - controller) 动态代理实现程序功能的统一维护技术

Code Coverage 代码覆盖率

① BeforeAll → ② BeforeEach → ③ Test → ④ AfterEach → ⑤ AfterAll
setup　　void.　　cleanup

Assertions　　@ Test
void … ( ) { assumeTrue("cz". equals (…)); }.

Design Pattern

Singleton Design Pattern - 保证对象只创建一个对象
(单例设计模式)

Provide various object creation
Creational Pattern mechanisms → 提高可扩展性,重用性

Composite Design Pattern.
Let you compose objects into tree structures to present part-whole hierarchy

Structural Pattern　Explain how to assemble objects and classes into larger structures, keeping the structures flexible and efficient.

Behavioral Pattern　Handle algorithm and the assignment of responsibilities between objects

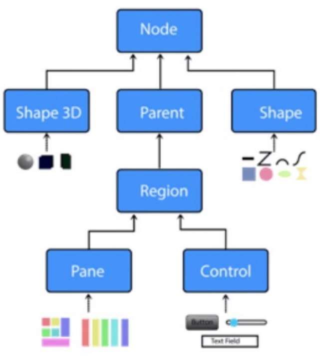
Command Design Pattern.
encapsulate a request into a standalone object that contains all information about the request.

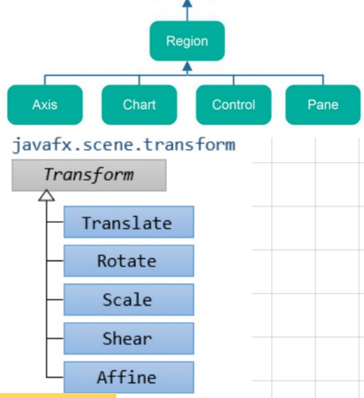
Decorder Pattern　a structural design pattern that allow behavior to be added to an individual object dynamically.

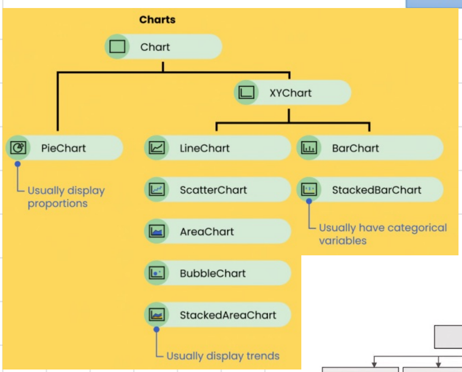
e.g. UML: Unified Modeling Language.

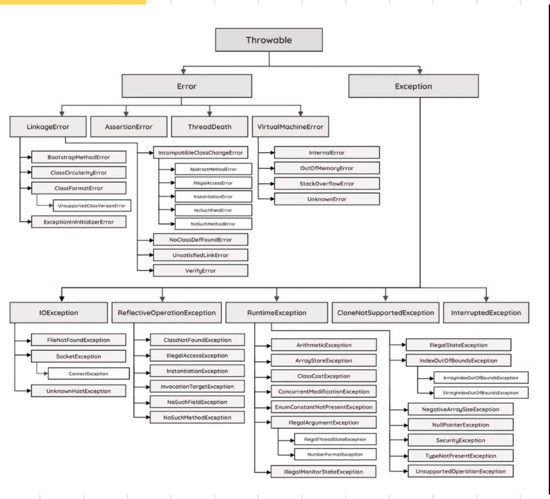
Array 不支持泛型 { why: 1. 类型擦除 2. 安全性

List [] arr = new List [10]
编译器无法知道 List 应该是什么类型
数组协变, 一个数组类型是另一个数组的子类
该数组类型安全, 有 ClassCastException 的发生.









1. Cohesion 和 Coupling 指的是什么？软件设计中，与之对应的原则是什么？
Modules (模块): A complex software system can be divided into simpler pieces called modules
Cohesion (内聚): How elements of a module are functionally related to each other
Coupling (耦合): How different modules depend on each other
High cohesion: modules are self-contained and have a single, well-defined purpose; all of its elements are directly related to the functionality that is meant to be provided by the module Low coupling: modules should be as independent as possible from other modules, so that changes to one module will have minimal impact on other modules
2. 为什么在使用泛型时不能使用原始类型（Primitive Type）而必须使用引用数据类型（Reference Type）？
Java中的泛型不是真正的泛型，而是语法糖。泛型在JVM的实现机制是擦除是作为Object存在，而基础数据类型没有继承自Object，会导致错误。而引用数据类型全部继承自Object，允许进行强制类型转换，不会引起错误。
3. Spring 中的 @Autowried 注解的保留注解（Retention Policy）是什么？ @Autowried 的原理是什么？
保留注解：RetentionPolicy.RUNTIME

source → complier 忽略. Source level
class → complier ✓ jvm 忽略.
runtime → jvm ✓

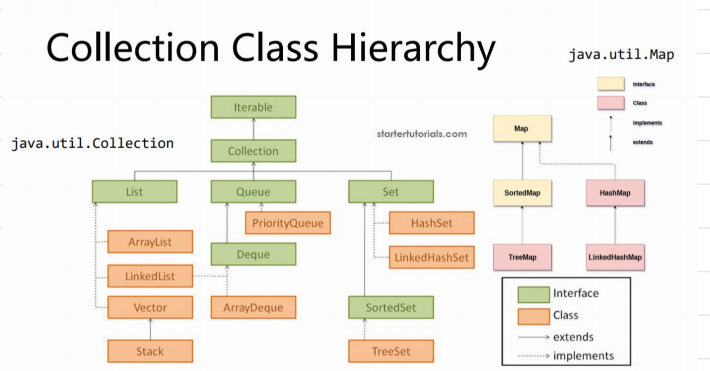
@Autowried 的原理：Autowiring allows the Spring container to automatically resolve dependencies between collaborating beans by inspecting the beans that have been configured
[RetentionPolicy.SOURCE/CLASS/RUNTIME]
4. Thread 本身就实现了 Runnable 的 run 方法，那为什么在多线程情境下，我们要调用 start 方法而不是 run 方法?
Thread 的 run 方法会直接调用我们传入的 Runnable 的 run 方法，也就是说，这只是一次普通的方法调用，并不会开启新线程。start方法才会开启新线程。

```
@override
public boolean equal ( object o ) {
    if (o instanceof CharPair ) {
        return ((CharPair) o).x == x && ((CharPair) o).y == y;
    } else {
        return false
    }
}
```

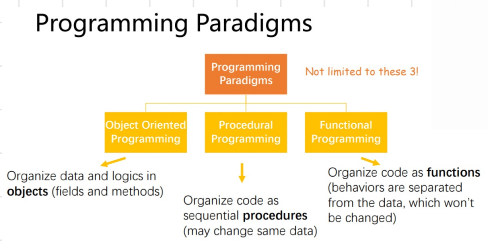
```
Stream.generate ( Math:: random)
    .limit (50). forEach ( system.out::println);
```

Collection → interface.
Collections → class

benifit of generics { Type safety / Type casting is not required / Complie-time checking



Type Erasure
the implementation of Java generics adopts the strategy of pseudo generics
Java supports generics in syntax, but the so -called "type erase" will be carried out in the compilation stage to replace all generic representations (contents in angle brackets) with specific types
To JVM, there is no generics at all,.



Interface	Function Signature	Example
UnaryOperator<T>	T apply(T t)	String::toLowerCase
BinaryOperator<T>	T apply(T t1, T t2)	BigInteger::add
Predicate<T>	boolean test(T t)	Collection::isEmpty
Function<T,R>	R apply(T t)	Arrays::asList
Supplier<T>	T get()	Instant::now
Consumer<T>	void accept(T t)	System.out::println

Lazy Evaluation　Benefits? 高效.

Intermediate operations are lazily executed
- They only remember the operations, but don't do anything right away (lazy)

Terminal operations are eagerly executed
- When terminal operations are initiated, the remembered operations are performed one by one (eager)