Java

**协议**：互相遵守的规则 **耦合**：层次的依赖性

**数据类型**

对象变量实际为引用类型（即地址, 值传递）。在 Java 中，任何对象变量的值都是对存储在另外一个地方的一个对象的引用。new 操作符的返回值也是一个引用。

|  |  |
| --- | --- |
| C++ | Java |
| Date \* pBirthday = new Date(); | Date birthday = new Date(); |
| delete pBirthday; | birthday = null; |

**类的外部访问控制符**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Modifier | Class | Package | Subclass | World |
| Public | Y | Y | Y | Y |
| Protected | Y | Y | Y | N |
| no modifier(package-private) | Y | Y | N | N |
| Private | Y | N | N | N |

The protected modifier specifies that the member can only be accessed within its own package (as with package-private) and, in addition, by a subclass of its class in another package.

**final**

1. 属性: 只读

2. 方法: 不可被重写（编译时绑定）

3. 类: 不可被继承

**static**

静态成员属于类的，只能访问静态成员，为类的所有对象共享。编译时绑定

对于静态变量在内存中只有一个拷贝，JVM只为静态分配一次内存，在加载类的过程中完成静态变量的内存分配，可用类名直接访问

public static void main(String[] args) {} //可作为类的单元测试入口

**interface(接口): 规范的定义（e.g. USB接口，驱动程序就是implements）**

规定一系列的（public static final）属性和（public abstract）**方法**的模型

**abstract class(抽象类):** 不可实例化，可实现部分方法

(interface -> abstract class -> class)

**Annotation(注解)**

Annotation是interface，关联1个RetentionPolicy，n个ElementType。

@Deprecated 废弃的

@Override 重写父类的方法

@SuppressWarnings 不提示WARNING

**Reflection(反射)**

获取类/对象的运行时信息

(Field/Method/Constructor/Modifier/Annotation)

*Generics*

**使用方式**

1. <T extends Comparable>上边界类型限定: assert(T instanceof Comparable);

2. <T super Integer>下边界类型限定: assert(Integer instanceof T);

3. <?> 一种特殊的未知类型实参通配符。? is a placeholder saying "I don't know or care what the generic type is" generally used when the work you'll do on the container object doesn't need to know the type.

**实现机制**

1. 编译时类型擦除: 删去类型参数，转为原始类型 (如不限定，则是Object)

2. 运行时强制类型转换: 原始类型，恢复类型参数

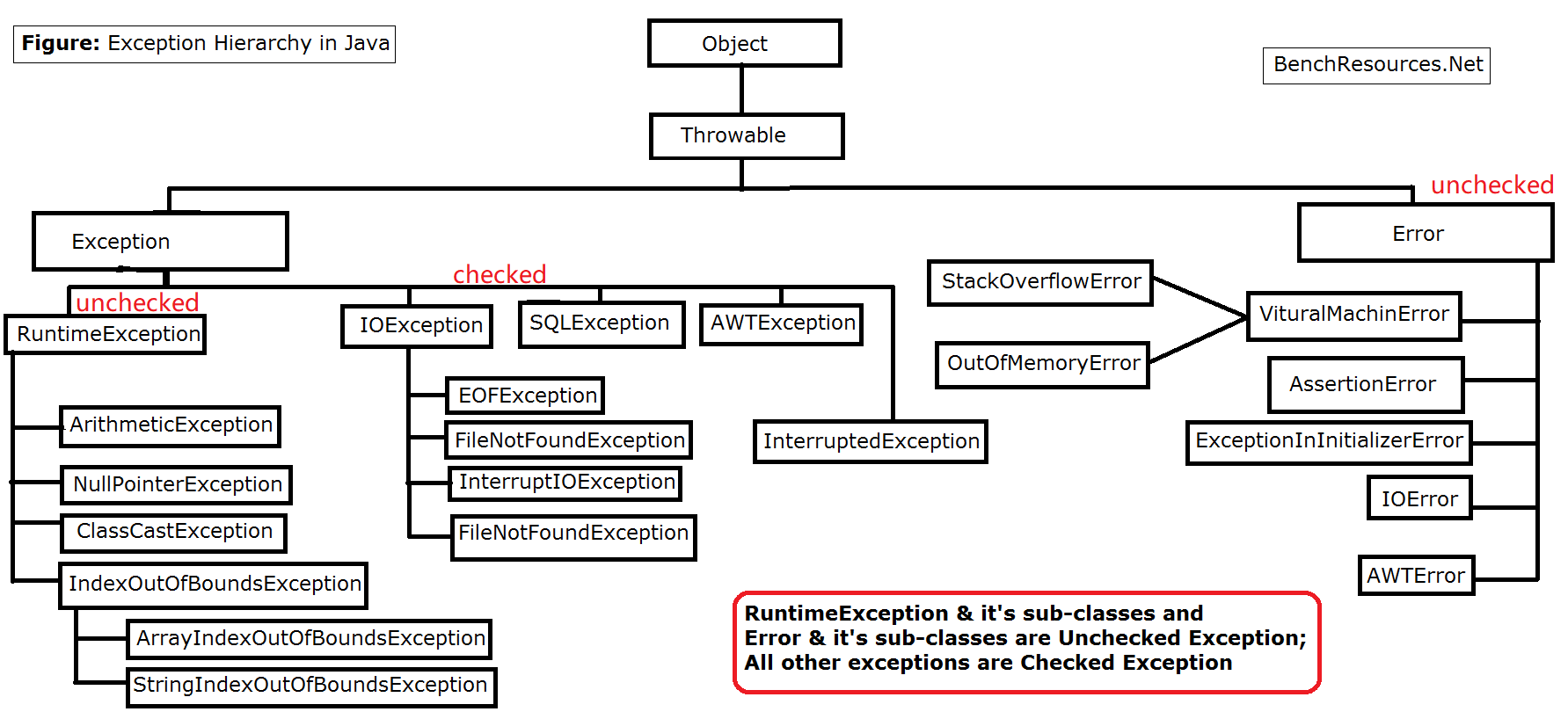
3. JVM没有泛型，只有普通的类和方法。所有的类型参数都用它们的限定类型替换。为了保证类型安全性，必要时加上强制类型转换。

*Exception*

try/catch/finally（可以解决if-else无能为力的情况），将错误处理与主干代码分开。

运行时的不希望发生的事件，就抛出特定类型的异常对象，若不捕获处理，则终止程序。

必须处理非RuntimeException的Exception类型异常。



throws… 表明函数潜在的异常交由调用者处理(若函数内部已处理异常，则不需throws)

**备注**

1. 异常机制花销大，不过分依赖。
2. 具体化异常层次，NumberFormatException extends IOException
3. 重写一个方法时，子类抛出的异常类型不能超出父类的异常类型的范围，否则无法用一个函数，对父类及其子类统一处理（继承的多态）。
4. 假定 A extends Exception; B extends A; C extends B;

则catch (C) {} catch(B) {} catch(A) {} catch(Exception) {}

【既然父类都出错了，那么基于父类的子类必出错】

*Object*

**public class Object**

protected Object clone() //Creates and returns a copy of this object.

String toString() // getClass().getName() + '@' + Integer.toHexString(hashCode()).

int hashCode() //Returns a hash code value for the object.

Class<?> getClass() //Returns the runtime class of this Object. reflection

boolean equals​(Object obj) // for any non-null reference values x and y, this method returns true if and only if x and y refer to the same object (x == y has the value true).

*Java Collections Framework*

**接口定义**



**通用实现**



**public interface Collection<E> extends Iterable<E>**

boolean add​(E e)

Iterator<E> iterator()

boolean addAll​(Collection<? extends E> c)

boolean contains​(Object o)

boolean isEmpty()

boolean remove​(Object o)

boolean retainAll​(Collection<?> c)// 求交集

int size()

void clear()

Object[] toArray()

**public interface Iterator<E> //不同于C++STL的iterator**

E next() //相当于元素之间的隔板，n个元素，能有(n+1)个插入隔板的位置

boolean hasNext()

//return true if next() would return an element rather than throwing an exception.

default void remove()

//IllegalStateException - if the next method has not yet been called, or the remove method has already been called after the last call to the next method

default void forEachRemaining​(Consumer<? super E> action) //lambda callback

**public interface Map<K,V>**

boolean containsKey​(Object key)

V put​(K key, V value)

//return the previous value associated with key, or null if there was no mapping for key.

V get​(Object key)

//return the value to which the specified key is mapped, or null if this map contains no mapping for the key

default VgetOrDefault​(Object key, V defaultValue)

V remove​(Object key)

default boolean remove​(Object key, Object value)

Set<Map.Entry<K,V>> entrySet()

Set<K>keySet()

Collection<V>values()

**public class Collections extends Object**

static <T> T max​(Collection<? extends T> coll, Comparator<? super T> comp)

static <T> void sort​(List<T> list, Comparator<? super T> c)

static void shuffle​(List<?> list)

static <T> int binarySearch​(List<? extends T> list, T key, Comparator<? super T> c)

static <T> boolean replaceAll​(List<T> list, T oldVal, T newVal)

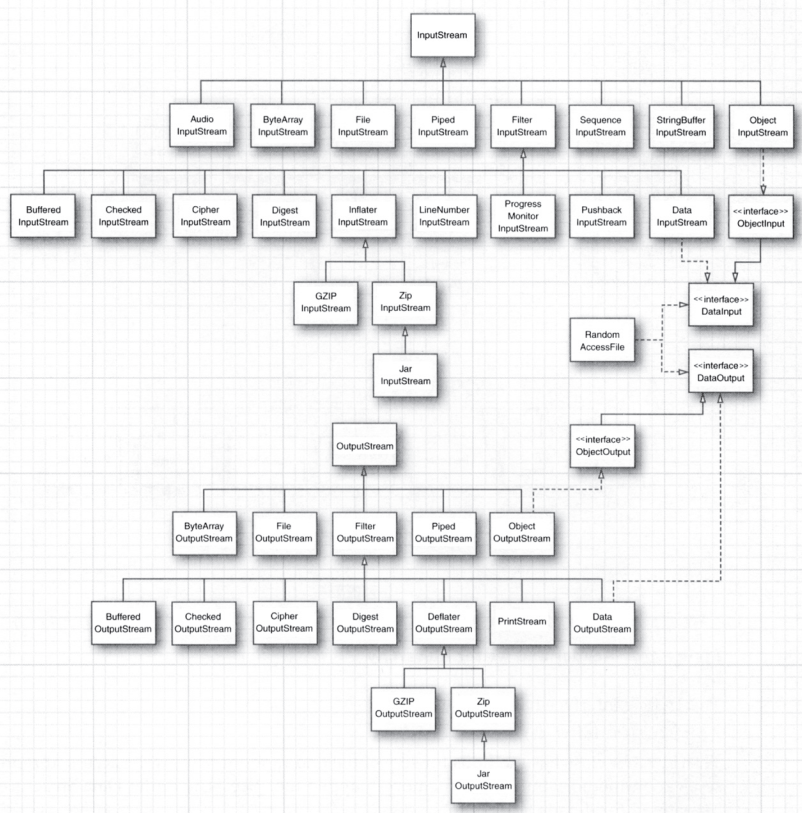
static <T> void copy​(List<? super T> dest, List<? extends T> src)

static <T> void fill​(List<? super T> list, T obj)

static int indexOfSubList​(List<?> source, List<?> target)

static void reverse​(List<?> list)

*IO*



|  |  |  |
| --- | --- | --- |
| **数据类型** | **读** | **写** |
| byte | InputStream | OutputStream |
| Unicode | Reader | Writer |

1. 流的decorator和pipe

2. 缓存化

3. 字符编码问题java.nio.charset.Charset

4. 对象序列化：Serializable内置的全序列化，默认static和transient不参与；

Externalizable定制的部分序列化。注意：敏感信息不参与序列化！

5. 流打开了，要及时关闭！GC只回收内存，不回收操作系统分配的资源。

*Lambda*

1. λ演算：无状态的计算表达式
2. stream()：流化迭代器
3. 常用运算子(ref. java.util.stream.Stream<T>)

|  |  |
| --- | --- |
| filter() | 过滤 |
| map() | 映射 |
| reduce() | 两两化简 |
| allMatch() | 是否全部匹配 |
| anyMatch() | 是否部分匹配 |
| noneMatch() | 是否无法匹配 |
| forEach() | 遍历 |

*Net*

AbstractPlainSocketImpl: Default Socket Implementation.

SocksSocketImpl: SOCKS (V4 & V5) TCP socket implementation (RFC 1928).