Java

使用类库：继承或调用方法

**协议**：互相遵守的规则 **耦合**：层次的依赖性

**garbage collection：**

动态分配的内存空间而没有被指向，回收没有被引用的内存，保证有引用的内存不被释放(null无指向)。**有了垃圾收集器并不意味着一定不会有内存泄漏。**

**基本数据类型：**

局部变量（函数内），必须初始化。当一个对象创建时，数据成员默认全为零。

对象变量实际为引用类型（即地址，Only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.

**类关系**

dependence：uses-a

aggregation：has-a

inheritance：is-a

**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

**构造函数：no destructor**

函数名为类名，不需写返回值类型.对象创建时，自动调用.

**（**若自定义构造函数，则编译器不会产生默认的构造函数**）**

区别：C++不允许数据成员定义时，初始化。而Java允许，随后执行构造函数。

**This：指向本身**

同一类的不同对象共用一个非静态方法，根据对象的this引用来操作.

**继承**

一个新类（派生类）从已有类（基类）获得属性和方法（除构造和析构外），子类可以继承父类的所有非私有的成员。

Java只支持单一继承，不允许多重继承（一个子类继承多个父类）。

overload:同函数名，不同的函数签名

override: 同函数名，相同的函数签名

**Super : 指向父类**

调用基类的构造函数，super(实参表); （派生类构造函数中第一条语句）

**多态：One接口，Multi实现**

父类的引用类型可以指向父类或子类，根据当前时刻的指向而采取不同的方法

Java语言全面支持动态绑定，而C++语言只对虚函数使用动态绑定

@Override在子类中重写只是隐藏了父类的方法, 权限不能变低

子类可以当成父类看待，但父类不是子类的一种

**final：**

-属性，只读

-方法，不可被重写（编译时绑定）

-类，不可被继承

**接口Interface：规范的定义（例如:USB接口，驱动程序就是implements）**

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

1.接口之间可以继承，甚至多继承

2.一个类实现接口的**部分**方法，则一定是抽象类，先extends后implements

3.Instanceof 判断是否包含接口的实现

4.由于使用某接口须实现所有方法，以通用实现类为拓展，重写特定方法，很有必要。

**抽象类：**不可实例化，**abstract，**可带有部分方法的实现

(interface->abstractclass->class)

**包Package：**.class文件以目录的形式分类，打包成jar

mygame.shared.Utilities相当于mygame/shared/Utilities.class

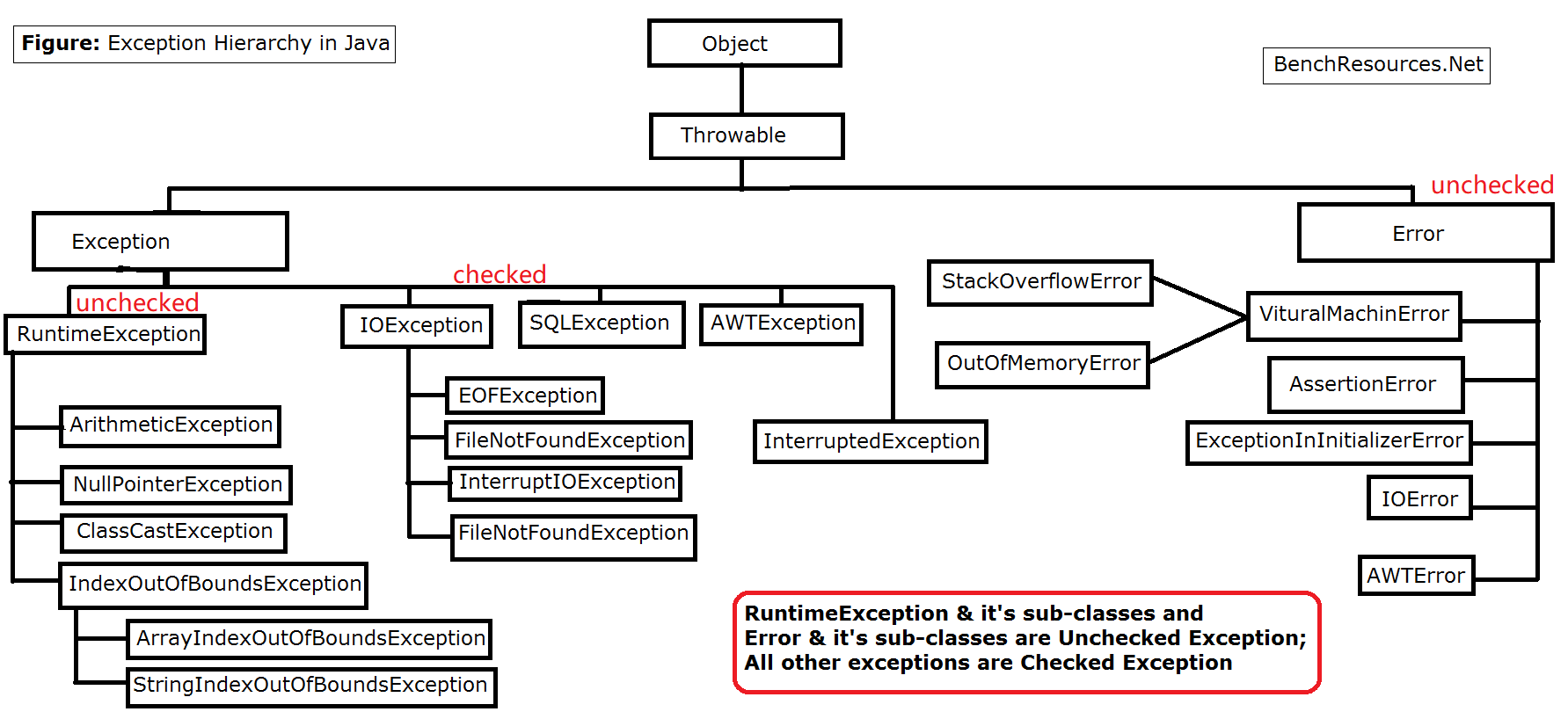
【javac –d . res.java 可以自动建立包层】

*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)

*Multithread: A thread is a thread of execution in a program.*

**Recommended usage：**

class Job implements Runnable {

@Override

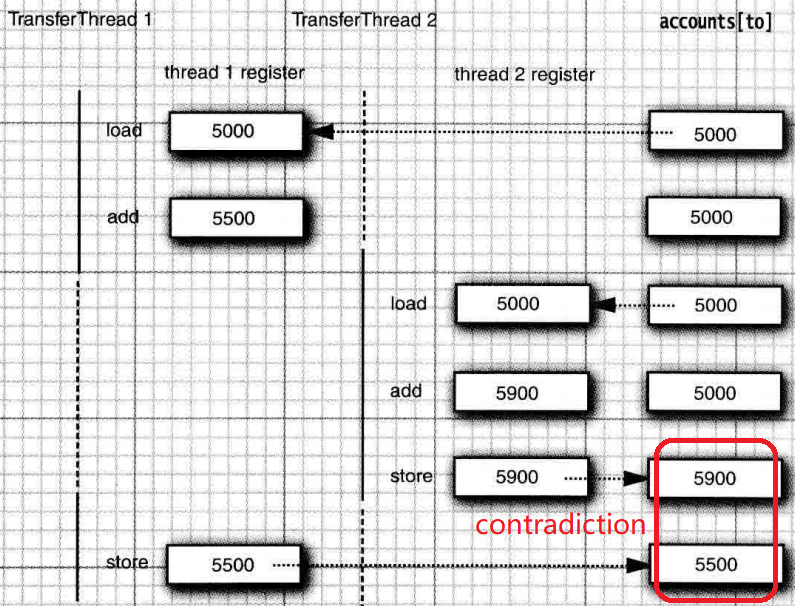
public void run() { //线程执行函数

// TODO

} //线程的终止：run()正常结束，或异常未捕获。

}

new Thread(new Job()).start();



抢占式调度系统给每一个可运行线程一个时间片来执行任务。当时间片用完，操作系统剥夺该线程的运行权，并给另一个线程运行机会。当选择下一个线程时，操作系统考虑线程的优先级。多处理器的机器上，多线程并行。但如果线程的数目大于处理器支持的数目，调度器依然采用时间片机制。

****

**Property:**

1. priority
2. daemon thread, 辅助用户线程，随时会结束。The Java Virtual Machine exits when the only threads running are all daemon threads.
3. UncaughtExceptionHandler
4. ThreadGroup

**Problem：**(万恶之源：多线程共享同一块存储空间)

共享且可变的数据/资源的竞争和一致性问题。

代码/过程的原子性和先后依赖问题。

**Solution:** (线程安全：多线程运行，不存在二义性，符合预期。)

1. Lock/synchronized --> resource(data/object) 【同步阻塞等待的思路】

任何时刻只有一个线程进入临界区。一旦一个线程封锁了锁对象，其他任何线程都无法通过lock语句。当其他线程调用lock时，它们被阻塞，直到第一个线程unlock锁对象。

FAULT: deadlock、starvation、livelock

**synchronized【个人，Lock】**

1. 修饰非static函数时，synchronized(this){}。锁定函数所在的对象。
2. 修饰static函数时，synchronized(.class){}。锁定函数所在的类。

**wait/notifyAll【团队，Condition】**

1. This method wait() causes the current thread to place itself in the wait set for this object and then to relinquish any and all synchronization claims on this object. 阻塞，解锁，等待激活通知。
2. The current thread must own this object's monitor lock. 因为wait()和notify()之间是通过对象的同步锁关联起来的。这也是notify(), wait()等函数定义在Object中，而不是Thread中的原因。
3. while (isOK(resource)) { // 避免重新激活后，资源的改变

object.wait();

}

1. 场景：线程间的协作、通信。