## JavaSE Foundation

1. 关于计算机系统中的数据表示：

* 位：bit(只有0，1两种状态)，是计算机系统中的最小数据表示单位。
* 字节：byte, 1 byte = 8 bit.(因为1个bit有两种状态，所以1 byte 可以表示 2的8次方个状态，即256个状态: -128~127)
* KB： 1KB = 1024 byte;
* MB: 1MB = 1024KB;
* ……..

1. 面向对象程序设计的三大基本特征：

* 继承(Inheritance)
* 封装(Encapsulation)：类包含了数据与方法，将数据与方法放在一个类中就构成了封装。

参数传递：Java的参数传递，无论传递的是原生数据类型还是引用类型，只有值传递(pass by value)。

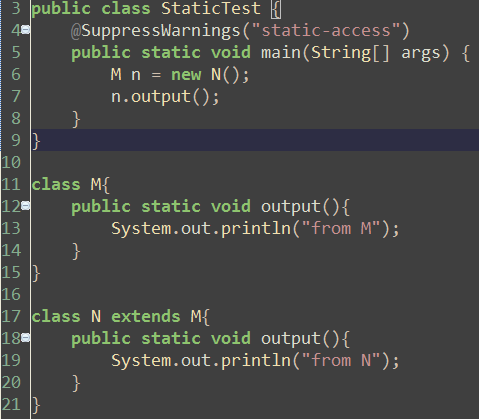
1. 对于原生类型，传的就是值，传过去后你是你，我是我，再没有关联；

对于引用类型，即是把引用的值传过去，引用的值也即是对象的地址，传递后两个引用指向的是同一个地址，任何一个引用对对象产生了改变，都会影响另一个引用。

* 多态(Polymorphism)：父类型的引用指向了子类型的对象，从而引起引用具有的属性和方法发生了变化。

引用的能力是由声明它的类型决定的，声明的为什么类型就具有什么样的方法，至于调用方法时表现出来的是声明类型的特征还是实现类型的，就决定于实现类型有没有重写其方法，引用不能调用子类拥有但声明类型不拥有的行为，除非先进行强类型转换。

1. static:



输出为”from M”

子类没有办法重写父类的方法，可以理解为父类把自己的static方法隐藏了。

静态方法只能继承，不可重写。

1. String s = “aaa”;这个语句做了什么事情？

有两种情况：

* 查找Strings pool中是否存在“aaa”这个对象，如果不存在，则在Strings Pool中创建一个“aaa”对象，然后将Strings Pool中创建好的这个对象的地址返回来，赋给引用变量s,这样s就指向了Strings Pool中的这个“aaa”字符串对象。
* 如果在Strings pool中存在“aaa”这个对象，则不创建任何对象，直接将“aaa”这个对象的地址返回给s

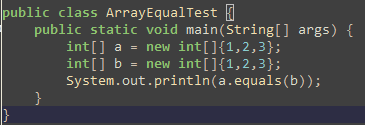
1. String s = new String(“aaa”)；

* 首先在Strings Pool中查找有没有“aaa”这个对象，如果有，则不在Strings Pool中再去创建“aaa”这个对象了，直接在堆中(heap)创建一个“aaa”对象的，然后将堆中的这个“aaa”对象的地址返回来，赋给s引用，导致s指向了堆中创建的这个“aaa”字符串对象
* 如果没有，则首先在Strings Pool中创建一个“aaa”对象，然后再在堆中创建一个“aaa”对象，然后将堆中的这个“aaa”对象的地址返回来，赋给s引用，导致s指向了堆中创建的这个“aaa”字符串对象

1. String的intern()方法功能：

|  |
| --- |
| **intern**  public [String](mk:@MSITStore:E:\Documents\Doc\JDK1.6.chm::/java/lang/String.html) **intern**()  Returns a canonical(典型的，规范的) representation for the string object.  A pool of strings, initially empty, is maintained privately by the class String.  When the intern method is invoked, if the pool already contains a string equal to this String object as determined by the [equals(Object)](mk:@MSITStore:E:\Documents\Doc\JDK1.6.chm::/java/lang/String.html#equals(java.lang.Object)) method, then the string from the pool is returned. Otherwise, this String object is added to the pool and a reference to this String object is returned.  It follows that for any two strings s and t, s.intern() == t.intern() is true if and only if s.equals(t) is true.  All literal strings and string-valued constant expressions are interned. String literals are defined in §3.10.5 of the [Java Language Specification](http://java.sun.com/docs/books/jls/html/)  **Returns:**  a string that has the same contents as this string, but is guaranteed to be **from a pool of unique strings**. |

1. Array equal test:



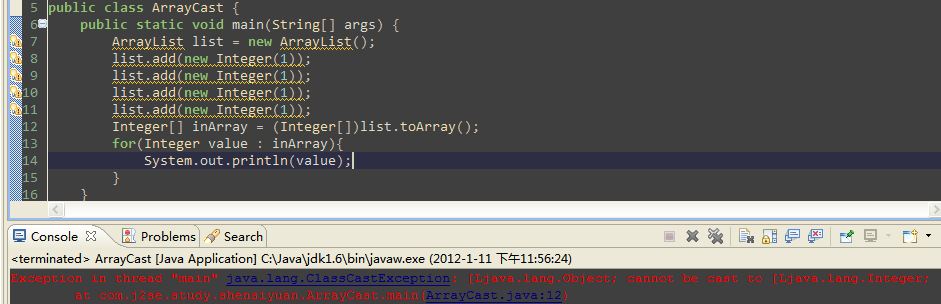
输出为false; 数组中的equal方法是比较地址的，如果真的要比较内容，则使用Arrays.equals(int[] a, int[]b)。

1. 拷贝数组可以使用System.arrayCopy(Object src, int srcPos, Object dest, int destPos, int len);
2. 冒泡排序、快速排序、交换排序和二分查找法。
3. 随机生成50个数字（整数），每个数字的范围是[10,50]，统计每个数字出现的次数以及出现次数最多的数字与它的个数，最后将每个数字及其出现次数打印出来，如果某个数字出现次数为0，则不要打印它。打印的时候按照数字的升序排列。
4. Eclipse有一个我们没有注意到但很有用的功能。



如上图当前光标在ENTER的后面，一般我们会按箭头把光标移动行尾，但其实不需要使用箭头，只需要在这个时候按一下回车键就自动跳到行尾了。

1. Object[]不能强制类型转换到Integer[]



1. 集合里放的和数组一样是对象的引用，而不是对象的本身。
2. Arraylist源代码分析：

Arraylist维护着一个叫做elementData的**Object**（**不是泛型类型的，它在get方法里先在内部强制类型转换再返回它的值**）数组、size属性，从AbstractList中继承过来的modCount属性，这个类中大量使用了Arrays.copyOf()和System.arrayCopy方法。

Arraylist底层采用数组实现，当使用不带参数的构造方法生成Arraylist对象时，实际上会在底层生成一个长度为10的Object数组，如果增加的元素个数超过了10个，那么Arraylist底层会新生成一个长度为原数组的1.5倍+1的数组。然后将原数组的内容复制到新数组当中，并且后续增加的内容都会放到新数组当中。当新数组无法容纳增加的元素时，重复这个过程。

对于Arraylist元素的删除操作，需要将被删除元素的后续元素向前移动，代码比较高。完成元素的移动后，并不会改变数组的长度，但会将最后一个“有值的”位置置为null，以便GC进行内存回收。

|  |
| --- |
| elementData[--size] = null; // Let gc do its work |

ensureCapacity方法代码如下：

|  |
| --- |
| /\*\*  \* Increases the capacity of this <tt>ArrayList</tt> instance, if  \* necessary, to ensure that it can hold at least the number of elements  \* specified by the minimum capacity argument.  \*  \* @param minCapacity the desired minimum capacity  \*/  public void ensureCapacity(int minCapacity) {  modCount++;  int oldCapacity = elementData.length;  if (minCapacity > oldCapacity) {  Object oldData[] = elementData;  int newCapacity = (oldCapacity \* 3)/2 + 1;  if (newCapacity < minCapacity)  newCapacity = minCapacity;  // minCapacity is usually close to size, so this is a win:  elementData = Arrays.copyOf(elementData, newCapacity);  }  } |

toArrays方法（不是直接返回elementData）

|  |
| --- |
| /\*\*  \* Returns an array containing all of the elements in this list  \* in proper sequence (from first to last element).  \*  \* <p>The returned array will be "safe" in that no references to it are  \* maintained by this list. (In other words, this method must allocate  \* a new array). The caller is thus free to modify the returned array.  \*  \* <p>This method acts as bridge between array-based and collection-based  \* APIs.  \*  \* @return an array containing all of the elements in this list in  \* proper sequence  \*/  public Object[] toArray() {  return Arrays.copyOf(elementData, size);  } |

1. Arraylist和Linkedlist的比较分析：

* Arraylist底层使用数组实现，Linkedlist底层采用双向链表实现。
* 当执行插入操作时，采用Linkedlist比较好。
* 当执行搜索操作时，采用Arraylist比较好。
* 当向Arraylist添加一个对象时，实际上就是将该对象放置到了Arraylist底层所维护的数组当中；当向Linkedlist中添加一个对象时，实际上LinkedList内部会生成一个Entry对象，该Entry对象的共有三个 属性： E element; Entry<E> next; Entry<E> previous;当中的element就是我们向LinkedList中所添加的元素，然后Entry又构造好了向前与向后的引用previous、next，最后将生成的这个Entry对象加入到了链表当中。**换句话说，LinkedList中所维护的是一个个的Entry对象，而不是我们添加的对象本身。**

1. 使用LinkedList来实现自己的Queue和Stack
2. Equals方法的API描述：

|  |
| --- |
| public boolean **equals**([Object](mk:@MSITStore:E:\Documents\Doc\JDK1.6.chm::/java/lang/Object.html) obj)  Indicates whether some other object is "equal to" this one.  The equals method implements an equivalence relation on non-null object references:   * It is *reflexive（自反性）*: for any non-null reference value x, x.equals(x) should return true. * It is *symmetric（对称性）*: for any non-null reference values x and y, x.equals(y) should return true if and only if y.equals(x) returns true. * It is *transitive（传递性）*: for any non-null reference values x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) should return true. * It is *consistent（一致性）*: for any non-null reference values x and y, multiple invocations of x.equals(y) consistently return true or consistently return false, provided no information used in equals comparisons on the objects is modified. * For any non-null reference value x, x.equals(null) should return false.   The equals method for class Object implements the most discriminating possible equivalence relation on objects; that is, 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).  Note that it is generally necessary to override the hashCode method whenever this method is overridden, so as to maintain the general contract for the hashCode method, which states that equal objects must have equal hash codes. |

hashCode

|  |
| --- |
| public int **hashCode**()  Returns a hash code value for the object. This method is supported for the benefit of hashtables such as those provided by java.util.Hashtable.  The general contract of hashCode is:   * Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently(一致地) return the same integer, provided no information used in equals comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application. 在java应用的一次执行过程当中，对于同一个对象的hashCode方法的多次调用，它们应当返回同样的值。（前提是该对象equals方法使用到的信息并没有发生变化），在java应用的多次执行过程中并不要求每次执行过程中返回的hashCode值要一致，即可以一致也可以不一致。 * If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result. 当equals方法返回true那么这两个对象的hashCode值必须是一致的。 * It is *not* required that if two objects are unequal according to the [equals(java.lang.Object)](mk:@MSITStore:E:\Documents\Doc\JDK1.6.chm::/java/lang/Object.html#equals(java.lang.Object)) method, then calling the hashCode method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hashtables. 两个对象如果它们的equals方法返回false,并不要求它们的hashCode一定要不一样，但对程序员而言，应当要了解到，如果不同则可以提高应用的性能。   As much as is reasonably practical, the hashCode method defined by class Object does return distinct integers for distinct objects. (This is typically implemented by converting the internal address of the object into an integer, but this implementation technique is not required by the JavaTM programming language.)  **Returns:**  a hash code value for this object.  **See Also:**  [equals(java.lang.Object)](mk:@MSITStore:E:\Documents\Doc\JDK1.6.chm::/java/lang/Object.html#equals(java.lang.Object)), [Hashtable](mk:@MSITStore:E:\Documents\Doc\JDK1.6.chm::/java/util/Hashtable.html) |

1. 所谓的字典顺序是按ASCII来说的，所以如果要按字典排序来对字母进行排序的话，小写字母要排在大写字母的前面，
2. Collections.reserveOrder();方法会返回一个comparator，所以要对列表进行反向排序有两种方法：

|  |
| --- |
| Arrays.sort(a, Collections.reverseOrder()); |

和

|  |
| --- |
| Collections.reservse(list); |

1. Map的keySet()方法会返回key的集合，因为Map的键是不可能重复的，所以返回的是Set类型，而Map的值是可以重复的，所以它的values()方法返回的就是Collection类型。
2. Main方法是可以传递参数的，如java test aa bb cc，那么这个main方法将有三个参数，默认会以空格来分割参数，但如果当要传递的参数里本来就带空格时，就可以用下来的方法来解决：java test “aa bb” cc，这时main方法就只有两个参数了。
3. 类里面可以再定义类，接口里面也可以再定义接口，如Map这个接口里就再定义了另一个接口Entry。
4. 遍历Map的常用两种方式：

|  |
| --- |
| Set<String> keySet = map.keySet();  for(String key : keySet){  System.out.println(key + "=" + map.get(key));  } |

和：

|  |
| --- |
| Set<Map.Entry<String, Integer>> entrySet = map.entrySet();  for(Map.Entry<String, Integer> entry : entrySet){  entry.getKey();  entry.getValue();  } |

1. Strategy Pattern

* 抽象策略角色
* 具体策略角色
* 环境类
* 客户端

抽象策略角色：

|  |
| --- |
| public interface Strategy {  int caculate(int a,int b);  } |

具体策略角色：

|  |
| --- |
| public class AddStrategy implements Strategy {  @Override  public int caculate(int a, int b) {  return a + b;  }  } |

具体策略角色：

|  |
| --- |
| public class MulStrategy implements Strategy {  @Override  public int caculate(int a, int b) {  return a \* b;  }  } |

环境类：

|  |
| --- |
| public class Enviroment {  private Strategy strategy;  public Enviroment(Strategy strategy) {  this.strategy = strategy;  }  public void **setStrategy**(Strategy strategy) {  this.strategy = strategy;  }  public int **caculate**(int a, int b){  return strategy.caculate(a, b);  }  } |

客户端：

|  |
| --- |
| public class Client {  public static void main(String[] args) {  Enviroment env = new Enviroment(new AddStrategy());  System.out.println(env.caculate(3, 4));    env.**setStrategy**(new MulStrategy());  System.out.println(env.caculate(3, 4));  }  } |

Strategy Pattern’s disadvantage:

* The client must know all the implemented strategy, and have to decide which strategy to use by itself.
* There will be many implemented strategy classes.

Solutions:

Use factory pattern.

1. HashSet底层是使用HashMap实现的。当使用Add方法将对象添加到Set中时，实际上是将该对象作为底层所维护的Map对象的Key，而Value则老师同一个Object对象PRESENT（这个对象我们用不上）
2. HashMap底层维护一个数组，我们向HashMap中所放置的对象实际上是存储在该数组当中；当向HashMap中put一对键值时，它会根据key的hashCode值计算出一个位置，该位置就是此对象准备往数组中存放的位置。如果该位置没有对象存在，就将此对象直接放到数组当中，如果该位置已经有对象存在了，则顺着此存在的对象的链开始寻找（Entry类有一个Entry类型的next成员变量，指向了该对象的下一个对象）。如果此链上有对象的话，再去使用equals方法进行比较，如果对此链上的某个对象的equals方法比较为false，则将该对象放到数组当中，然后将数组中该位置以前存在的那个对象链接到此对象的后面。
3. 下面的代码是否能编译通过？

|  |
| --- |
| GenericFoo<Integer> foo1 = new GenericFoo<Integer>();  foo1.setFoo(20);  String b = (String)foo1.getFoo(); |

不能。类型转换只能在不知道被转型类型是什么类型的情况下发生，如果编译时已经知道被转型对象的具体类型，则不会通过编译。

1. 在定义泛型类别时，预设可以使用任何的类型来实例化泛型类型中的类型，但是如果想要限制使用泛型类别时，只能 用某个特定类型或者其子类型才能实例化该类时，可以在定义类型时，使用extends关键字指定这个类型必须是继承某个类，或者实现某个接口。如：Class MyGeneric<T extends List> {}，当没有指定泛型继承的类型或者接口时，默认为T extends Object。
2. GenericTest< ? extends List> ge = null;这个语句声明了ge这个引用可以指向如下的类型：

Ge = new GenericTest<ArrayList>();也可以指向: ge = new GenericTest<LinkedList>();(这个时候类的声明为：class GenericTest<T>{})

1. 28使用的方式是在类声明的时候就指定实例化的类型必须是List的实现类，而29这种方式是在使用该类的时候才指定该引用的类型。
2. 使用<?>或是<? Extends SomeClass>的声明方式，意味着您只能通过该名称来取得所参考实例的信息，或者是移除某些信息，但不能增加它的信息，因为只知道当中放置的是SomeClass的子类，但不确定是什么类的实例，编译器不让您加入信息，理由是，如果可以加入信息的话，那么您就得刻取回的实例是什么类型，然后转换为原来的类型方或进行操作，这就失去了使用泛型的意义。

|  |
| --- |
| GenericTest<String> ge1 = **new** GenericTest<String>();  ge1.setFoo("hello");  GenericTest<? **extends** Object> ge2 = ge1;  System.*out*.println(ge2);  ge2.setFoo(**null**);  System.*out*.println(ge2);  //ge2.setFoo("welcome"); |

上面注释的代码会引起编译错误。

1. 一般而言，开发者社区说到动态语言，大致认同的一个定义是：“程序运行时，允许改变程序结构或变量类型，这种语言称为动态语言”。从这个观点上看，Perl，Python，Ruby，JavaScript是动态语言，C++，Java，C#不是动态语言。
2. 获取某个类或某个对象所对应的Classic对象的常用的3种方式：

* 使用Class类的静态方法forName：Class.forName(“java.lang.String”);
* 使用类的.class方法：String.class
* 使用对象的getClass()方法：String s = “aa”; Class<?> clazz = s.getClass();