java基础：编译时和运行时的区别

徐工
爱好广泛，最爱扯淡

在java开发设计过程中，了解java运行时和编译时的区别是非常有必要的。如下从几个问题来描述两者的区别

Q1: 如下代码片段中，A行和B行的区别是什么

```
public class ConstantFolding {
    static final  int number1 = 5;
    static final  int number2 = 6;
    static int number3 = 5;
    static int number4= 6;

    public static void main(String[ ] args) {
          int product1 = number1 * number2;             //line A
          int product2 = number3 * number4;             //line B
    }
}
```

A行是在编译时计算值，B行是在运行时计算值，当该类编译后，如果使用一些反编译器(如jd-gui)反编译后可以看到，实际代码如下：

```
public class ConstantFolding
{
  static final int number1 = 5;
  static final int number2 = 6;
  static int number3 = 5;
  static int number4 = 6;

  public static void main(String[ ] args)
  {
      int product1 = 30;
      int product2 = number3 * number4;
  }
}
```

java编译时会做一些优化操作，比如替换一些final的不可变更的参数，在这里，由于number1和number2都是final的，那么product1肯定是确定的，这里就会在编译时计算出product1的值。

除了如上的一些代码优化话，再什么其他的情况下查看编译后的class文件是非常有用的？

java中的泛型。泛型是编译时会做优化，通过编译文件可以非常方便的看到其对应的实际类型，如下例子：

实际编码如下：

```java
public class GenericsInspect {

    public static void main(String[] args){
        Parent p1 = new Dad();
        System.out.println(p1.getName());
        Parent p2 = new Mom();
        System.out.println(p2.getName());
    }
    private static interface Parent{String getName();}
    private static class Dad implements Parent{
        public String getName(){return "Dad...";}
    }
    private static class Mom implements Parent{
        public String getName(){return "Mom...";}
    }
}
```

反编译后的代码如下：

```java
public class GenericsInspect {
    public GenericsInspect() {
    }

    public static void main(String[] args) {
        GenericsInspect.Dad p1 = new GenericsInspect.Dad(null);
        System.out.println(p1.getName());
        GenericsInspect.Mom p2 = new GenericsInspect.Mom(null);
        System.out.println(p2.getName());
    }

    private static class Mom implements GenericsInspect.Parent {
        private Mom() {
        }

        public String getName() { return "Mom"; }
    }

    private static class Dad implements GenericsInspect.Parent {
        private Dad() {
        }

        public String getName() { return "Dad"; }
    }

    private interface Parent {
```

可以，在编译后的文件中，Parent类会显示的被实际类型取代。

重写，重载，泛型，分别是在运行时还是编译时执行的

1. 方法重载是在编译时执行的，因为，在编译的时候，如果调用了一个重载的方法，那么编译时必须确定他调用的方法是哪个。如：

```java
public class {
     public static void evaluate(String param1);  // method #1
     public static void evaluate(int param1);     // method #2
}
```

当调用evaluate("hello")时候，我们在编译时就可以确定他调用的method #1.

2. 方法的重写是在运行时进行的。这个也常被称为运行时多态的体现。编译器是没有办法知道它调用的到底是那个方法，相反的，只有在jvm执行过程中，才知晓到底是父子类中的哪个方法被调用了。如下：

```
public class A {
    public int compute(int input) {          //method #3
        return 3 * input;
    }
}

public class B extends A {
    @Override
    public int compute(int input) {          //method #4
        return 4 * input;
    }
}
```

试想，当有如下一个接口的时候，我们是无法确定到底是调用父类还是子类的方法

```
public int evaluate(A reference, int arg2)  {
     int result = reference.compute(arg2);
}
```

3. 泛型(类型检测)，这个发生在编译时。编译器会在编译时对泛型类型进行检测，并吧他重写成实际的对象类型(非泛型代码)，这样就可以被JVM执行了。这个过程被称为"类型擦除"。
类型擦除的关键在于从泛型类型中清除类型参数的相关信息，并且再必要的时候添加类型检查和类型转换的方法。

类型擦除可以简单的理解为将泛型java代码转换为普通java代码，只不过编译器更直接点，将泛型java代码直接转换成普通java字节码。类型擦除的主要过程如下：
1). 将所有的泛型参数用其最左边界（最顶级的父类型）类型替换。
2). 移除所有的类型参数。

```
List<String> myList = new ArrayList<String>(10);
```

在编译后变成：

```
List myList = new ArrayList(10);
```

4. 注解。注解即有可能是运行时也有可能是编译时。
如java中的@Override注解就是典型的编译时注解，他会在编译时会检查一些简单的如拼写的错误(与父类方法不相同)等

同样的@Test注解是junit框架的注解，他是一个运行时注解，他可以在运行时动态的配置相关信息如timeout等。

5. 异常。异常即有可能是运行时异常，也有可能是编译时异常。

RuntimeException是一个用于指示编译器不需要检查的异常。RuntimeException 是在jvm运行过程中抛出异常的父类。对于运行时异常是不需要再方法中显示的捕获或者处理的，如 *NullPointerException*, *ArrayIndexOutOfBoundsException*

已检查的异常是被编译器在编译时候已经检查过的异常，这些异常需要在try/catch块中处理的异常。

6. AOP. Aspects能够在编译时，预编译时以及运行时使用。

1). 编译时：当你拥有源码的时候，AOP编译器(AspectJ编译器)能够编译源码并生成编织后的class。这些编织进入的额外功能是在编译时放进去的。

2). 预编译时：织入过程有时候也叫二进制织入，它是用来织入到哪些已经存在的class文件或者jar中的。

3). 运行时：当被织入的对象已经被加载如jvm中后，可以动态的织入到这些类中一些信息。

7. 继承：继承是编译时执行的，它是静态的。这个过程编译后就已经确定

8. 代理(delegate)：也称动态代理，是在运行时执行。

你如何理解"组合优于继承"这句话

继承是一个多态的工具，而非重用工具。在没有多态关联关系的对象间，一些程序员倾向于使用继承来保持重用。但事实是，只有当子类和父类的关系为"is a"的关系时候，继承才会使用。

1. 不要使用继承来实现代码的重用。如果两者之间没有"is a"的关系，那么使用组合来实现重用。当父类的某个方法修改后，子类的相关实现也有可能会被更改。

2. 不要为了多态而使用继承。如果你只是为了实现多态而采用继承模式，那么实际上组合模式更加适合你，而且更加简洁和灵活。

这也就是为什么GoF设计模式中常说"组合优于继承"的原因。

你能区分编译时继承和运行时继承的区别吗？请列举例子说明

实际上在java中只支持编译时继承。java语言原生是不支持运行时时继承的。一般情况下所谓编译时继承如下：

```
public class Parent {
    public String saySomething( ) {
          return “Parent is called”;
    }
}
```

```
public class Child extends Parent {
     @Override
     public String saySomething( ) {
          return super.saySomething( ) +  “, Child is called”;
     }
}
```

如上有两个类，其中Child为Parent的子类。当我们创建一个Parent实例的时候(无论实际对象为Parent还是Child)，编译器在编译期间会将其替换成实际类型。所以继承实际上在编译时就已经确定了。
而在java中，可以设计通过组合模式来尝试模拟下所谓的运行时继承。

```
public class Parent {
    public String saySomething( ) {
          return “Parent is called”;
    }
}
```

```
public class Child  {
     private Parent parent = null;

     public Child( ){
          this.parent = new Parent( );
     }

     public String saySomething( ) {
          return this.parent.saySomething( ) +  “, Child is called”;
    }
}
```

在Child类中，其中有一个Parent实例。通过这种方式，我们动态的child类中代理了parent的相关功能。