CIS 410/510 : Principles of Object-Oriented Programming
Assignment 4 - Solutions

Question 1
1. Static error since zero has type Zero and in class Zero there is no m method.

2. output 2. Indeed, casting zero (which has dynamic type Two) to the supertype One is correct at run-time. Casting does not affect the dynamic type, hence the version of m in Two is executed.

3. output 0. Indeed, class One has two f methods, one inherited from Zero with parameter type Zero and one with parameter type One. Since zero has type Zero, the only applicable method is the former.

4. output 2. Indeed in this case, by the same argument as above, due to the cast the most specific applicable method f is that declared in One, whose effect is to invoke m on the argument; since the argument has dynamic type Two, the version of m in Two is executed.

5. Run-time error. Indeed at run-time casting one (which has dynamic type One) to a subtype raises an error.

Question 2
1. The former method declaration overrides that in A, whereas the latter defines a different method with the same name (overloading).

2. 

<table>
<thead>
<tr>
<th></th>
<th>aa</th>
<th>ab</th>
<th>bb</th>
</tr>
</thead>
<tbody>
<tr>
<td>aa</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ab</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>bb</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

3. class A { int bin(A x){ return 1; } }
class B extends A{
    int bin (A x){
        if (x instanceof B) return bin((B)(x)); else return super.bin(x); }
    int bin (B x){ return 2;}
}

Question 3
class Break {
    int m (Parent p) { return p.f.height;}
}

Heir h = new Heir(); h.f = new Rectangle(); new Break().m(h);

Question 4
1. The expression is not well-typed in Java, since new Heir().m() turns out to be of type Parent, and no method m can be found in Parent (a down cast should be inserted). Referring to the MiniJava typing rules, and denoting by Γ_P the class type environment extracted from the program P consisting of the two classes:

\[
\frac{\Gamma_P \vdash \text{Heir : class}}{(\Gamma_P, \emptyset) \vdash \text{new Heir: Heir}}\quad \frac{\Gamma_P \vdash \text{Heir}.m : \text{Parent}}{(\Gamma_P, \emptyset) \vdash \text{new Heir.m()} : \text{Parent}}
\]
At this point, the only typing rule which could be applied to the expression is (Method call), but this rule cannot be applied since the class type of Parent in \( P \) does not include a method named \( n \).

2. No. The evaluation would give 2. Indeed the evaluation of \( \text{new Heir().m()} \) returns an object identifier of class Heir, and at run-time the method \( n \) is found in class Heir class and executed. This can be formally shown using MINIJAVA reduction rules as follows:

\[
\begin{align*}
\text{Step 1:} & \quad (\text{new Heir,} \emptyset) \sim_p (\text{Heir,} \sigma) \\
\text{Step 2:} & \quad (\text{Heir.m(),} \emptyset) \sim_p (\text{Heir,} \sigma) \\
\text{Step 3:} & \quad (\text{Heir.n(),} \sigma) \sim_p (\text{2,} \sigma)
\end{align*}
\]

3. The problem is that the Java type system is in this case too restrictive. In a language allowing covariant change in return type in method overriding, we could write:

```java
class Heir extends Parent {
    Heir m () { return this; }
    int n () { return 2; }
}
```

or, even better, in a language with MyType we could write:

```java
class Parent { MyType m () { return this; }
```

Question 5

1. The assignment is ill-typed in Java since Client2 is not a subtype of Client1, being the subtyping relation determined by the inheritance relation.

2. ParentType = ObjectType {} 
   HeirType = ObjectType { m : Void -> Integer; } 
   Client1Type = ObjectType { m : HeirType -> Integer; n : Void -> ParentType } 
   Client2Type = ObjectType { m : ParentType -> Integer; n : Void -> HeirType } 
   class Parent {} 
   class Heir inherits Parent { 
       function m () : Integer is { return 1 } 
   } 
   class Client1 { 
       f1 : Integer := 0; 
       function m (x : HeirType) : Integer is { return x <= m() } 
       function n () : ParentType is { return new Parent } 
   } 
   class Client2 { 
       f2 : Integer := 0; 
       function m (x : ParentType) : Integer is { return 2 } 
       function n () : HeirType is { return new Heir } 
   } 

3. The equivalent of the previous assignment is \( c1=\text{new Client2} \) with \( c1 \) of type Client1Type. This assignment is well-typed in SOQL since Client2Type turns out to be a subtype of Client1Type; indeed HeirType is clearly a subtype of ParentType and Client2Type has the same methods of Client1Type with more specific return type and more generic parameter type.