SWEN 423 Term Test 1

Released 8am Friday 7 August Due 8am Saturday 9 August NZ time.

Total 120 points, capped at 100; so if you get 105 or 110 points is exactly like getting 100%.

Here we consider a version of FJ with exceptions, called EFJ

Section 1: Grammar

```
e ::= x | e.m(es)  | new C(es) | e.f | try{e}catch(C x){e'} | throw e
cd ::= class C implements Cs{ Fs K Ms }
 | interface C extends Cs{ MH₁; .. MHₖ; }
K ::= C(C₁ x₁ .. Cₙ xₙ){this.x₁=x₁;..this.xₙ=xₙ;}
F ::= C x;
M ::= MH{return e;}
MH ::= C m(C₁ x₁ .. Cₙ xₙ) throws Cs
v ::= new C(vs)
Ɛv ::= □ | Ɛv.m(es) | v.m(vs,Ɛv,es)| new C(vs,Ɛv,es) | Ɛv.x | throw Ɛv
Γ ::= x₁ : C₁ .. xₙ : Cₙ
```

As for implements and extends, in Java when the "throws" declarations is empty, also the "throws" keyword is omitted, while in EFJ we would just write the empty list of Cs.
As you can see, try-catch is now an expression. This is different w.r.t. Java, where try-catch is a statement.

That is,
`try{a.m()}catch(X x){y}`
is an expression, that will evaluate in the result of `a.m()` (if no exception is leaked)
or into `y` (if an exception of dynamic type `X` is leaked by the execution of `a.m()`)
or it will propagate the exception leaked by the execution of `a.m()` if such exception is not of type `X`.

Also in EFJ we do not have a "root of all exceptions", but all instance can be thrown as exceptions.

Q1[10 marks]: Consider the following well typed Java program:

```
class A extends Throwable{}
class B extends Throwable{}
class User{
  A m1()throws A{throw new A();}
  A m2(){
    A a;
    try{a=m1();}catch(A x){a=x;}
    return a;
    }
  }
```

This program is not a valid EFJ program. Write an equivalent program in EFJ.

Section 2: Reduction

$$\frac{e_1 \rightarrow e_2}{\mathcal{E}v[e_1] \rightarrow \mathcal{E}v[e_2]}\ (ctx)$$

Q2[10 marks]:
Write the rule to reduce field access (f-access). Do you think it is influenced by exception handling?

$$\frac{\text{class } C \ \_\{\ \_\ C_0\ m(C_1\ x_1..C_n\ x_n)\{\text{return } e;\}\ \_\ \}\ \text{in cds}}{\text{new } C(vs).m(v_1..v_n) \rightarrow e[\text{this}=\text{new } C(vs), x_1=v_1..x_n=v_n]}\ (m\text{-}call)$$

$$\frac{C \leq C'}{\text{try}\{\mathcal{E}v[\text{throw new } C(vs)]\}\text{catch}(C'\ x)\{e\} \rightarrow e[x=\text{new } C(vs)]}\ (try\text{-}catch)$$

Q3: describe in detail how rule (try-catch) is working:
Q3a[5 marks]:-What exactly "$\mathcal{E}v[\text{throw new } C(vs)]$" means?
Q3b[5 marks]:-How is this rule interacting with the carefully declared definition of $\mathcal{E}v$?
Q3c[5 marks]:-We have to call explicitly the subtyping relation. Why can we not just have $C=C'$

$$\frac{\text{not } C \leq C'}{\text{try}\{\mathcal{E}v[\text{throw new } C(vs)]\}\text{catch}(C'\ x)\{e\} \rightarrow \text{throw new } C(vs)}\ (try\text{-}miss)$$

Q4[10 marks]: we still need to add exactly 1 rule, to handle the case where there is no exception leaked by the try expression. Add such rule.

Section 3: Expression Type System

$$\frac{}{\Gamma;Cs \vdash x : \Gamma(x)}\ (x\text{-}t)$$

$$\frac{\Gamma;Cs \vdash e_0 : C_0 \;..\; \Gamma;Cs \vdash e_n : C_n \quad \_ \; C_0 \; \_ \; \{\_ \; C \; m(C_1\; x_1..C_n\; x_n) \_ \; \_\} \text{ in } cds}{\Gamma;Cs \vdash e_0.m(e_1\; ..\; e_n) : C}\ (m\text{-}t)$$

Q5[10 marks]: rule (m-t) is wrong because is incomplete. It need to check something related to exceptions. Please write the correct version of (m-t)

$$\frac{\Gamma;Cs \vdash e_1 : C_1 \;..\; \Gamma;Cs \vdash e : C_n \quad \text{class } C \; \_ \; \{C_1\; x_1;..C_n\; x_n;\; K\} \text{ in } cds}{\Gamma;Cs \vdash \text{new } C(e_1\; ..\; e_n) : T}\ (n\text{-}t)$$

Q6[5 marks]: Find the 3 obvious typos in rule (n-t).

$$\frac{\text{class } C \; \_ \; \{C_1\; x_1;..C_n\; x_n;\; K\; Ms\} \text{ in } cds \quad \Gamma;Cs \vdash e : C}{\Gamma;Cs \vdash e.x_i : C_i}\ (f\text{-}t)$$

$$\frac{\Gamma;Cs\ C \vdash e: C \quad \Gamma\ x{:}C;Cs \vdash e': C}{\Gamma;Cs \vdash \text{try}\{e\}\text{catch}(C\ x)\{e'\} : C}\ (\text{try-catch-t})$$

Q7[5 marks]: describe rule (try-catch-t) in English. Use correct terminology.

$$\frac{\Gamma;Cs \vdash e: C_0 \quad C_0 \text{ in } Cs}{\Gamma;Cs \vdash \text{throw } e : C_1}\ (\text{throw-t})$$

Q8[10 marks]: here we just check C_0 in Cs. However, in rule (try-catch) and (try-miss) we check for subtyping $C \leq C'$ explicitly.
Can you explain why this is needed in rules (try-catch) and (try-miss) but is not needed in (throw-t)? be sure to include the name of any other rule that relevant to allow this simplification.

Q9[10 marks]:
we type 'e' in 'C_0', but 'throw e' in 'C_1', and there is no relation between 'C_0' and 'C_1'.
Is this a mistake? Is this correct? Justify your answer.

Section 4: Class Type System and Subtyping

$$
\text{(meth)}\ \frac{\begin{array}{l}\texttt{overrideOk(C',}C_0\ \texttt{m(}C_1\ x_1..C_n\ x_n\texttt{))}\ \texttt{forall C' in Cs}\\ \texttt{this:C}\ x_1\texttt{:}C_1..x_n\texttt{:}C_n\texttt{; Cs' RunTimeException Error |- e :}\ C_0\end{array}}{\texttt{C;Cs |-}\ C_0\ \texttt{m(}C_1\ x_1..C_n\ x_n\texttt{)throws Cs'\{return e;\} : ok}}
$$

Q10[10 marks]: Rule (meth) uses the terminals `RunTimeException` and `Error`; those are just constants of form C.
What is the impact of using them? Use correct terminology, we expect at least two paragraph.

$$
\text{(id)}\ \frac{\begin{array}{l}\texttt{overrideOk(C',}MH_1\texttt{) forall C' in Cs}\\ \texttt{..}\\ \texttt{overrideOk(C',}MH_n\texttt{) forall C' in Cs}\\ \texttt{dom(C)} \subseteq \texttt{dom(C') forall C' in Cs}\end{array}}{\texttt{interface C extends Cs\{}MH_1\texttt{;..}MH_n\texttt{;\} : ok}}
$$

$$
\text{(cd)}\ \frac{\begin{array}{l}\texttt{C;Cs |-}\ M_1\texttt{: ok .. C;Cs |-}\ M_n\texttt{: ok}\\ \texttt{dom(C)} \subseteq \texttt{dom(C') forall C' in Cs}\end{array}}{\texttt{class C implements Cs\{ Fs K}\ M_1..M_n\texttt{\}:ok}}
$$

Q11[5 marks]: There are two obvious typos in the rule (id) and two similar ones in rule (cs).
Identify those typos.

In the following you can find conventional subsumption and subtyping rules.

$$
\text{(sub)}\ \frac{\begin{array}{l}C \leq C'\\ \Gamma \vdash e : C\end{array}}{\Gamma \vdash e : C'}
$$

$$
\text{(sub-direct)}\ \frac{\begin{array}{l}\_\ C_0\ \_\ C_1..C_n\ \{\_\}\ \text{in cds}\\ i\ \text{in}\ 0..n\end{array}}{C_0 \leq C_i}
$$

$$
\text{(sub-trans)}\ \frac{\begin{array}{l}C_1 \leq C_2\\ C_2 \leq C_3\end{array}}{C_1 \leq C_3}
$$

Section 5: Other notations

```
_______
#Define e₀[x=e₁] = e₂
 x[x=e] = e
 x[x'=e] = x where: x != x'
 e₀.m(e₁..eₙ)[x=e] = e₀[x=e].m(e₁[x=e]..eₙ[x=e])
 new C(e₁..eₙ)[x=e] = new C(e₁[x=e]..eₙ[x=e])
 e₀.f[x=e] = e₀[x=e].f
```

```
_______
#Define overrideOk(C,MH)
```

Q12[5 marks]: define `overrideOk(C,MH)`. Does something changes with respect to FJ?

```
_______
#Define dom(C)
 dom(C) = m₁..mₙ
 class C _{Fs K M₁..Mₙ} in cds
 Mᵢ= _ mᵢ(_){_}
 dom(C) = m₁..mₙ
 interface C _{MH₁..MHₙ} in cd
```

Q13[15 marks]: Write all the steps of the small step reduction for the following main expression e in the following class table cds:

```
cds =
  class Error{}
  class A{ A meth() throws B {return throw new B();} }
  class B{}
  class C{
    B m1() throws B {return try{new A().meth()}catch(A a){a};}
    B m2(){try{return this.m1()}catch(B b){b};}
    }

e = try{new C().m2()}catch(C c){throw new Error()}
```