Question 1 - Definition of new reduction and typing rules Assume we extend MINIJAVA with cast and instanceof operators as in Java, with the following syntax

\[ e ::= (C) e | e \text{ instanceof } C \]

and the standard meaning (see course notes 5.2).
Add reduction and typing rules for the operators. You should add 4+4 reduction rules and 1+1 typing rules. For the reduction rules, add to run-time expressions the new error \texttt{castExc} which is obtained when a cast fails at run-time. Note also that the instanceof operator fails on a null argument.

Question 2 - Implementation of recursive data structures with error handling in Java Consider the following Java abstract class which implements sets of integer numbers, where the method isin tests whether a number belongs to the receiver, the method cancelElem returns a (new!) set whose elements are the same of the receiver except for the argument, the method min returns the minimum element and throws EmptySetException when the receiver is empty.

abstract class Set {
  abstract boolean isin (int e);
  abstract int min () throws EmptySetException ;
  abstract Set cancelElem (int e) ;
}
class EmptySetException extends Exception {}

- Define two heir classes Interval and Union (with suitable constructors) which implement, respectively:
  - sets which are intervals of the form \([a, b]\) (empty if \(b < a\)),
  - sets obtained as union of two given sets.
- Add in class Set a static method which returns the difference between two given sets \(s1\) and \(s2\) (that is, the set whose elements are all the elements of \(s1\) which do not belong to \(s2\)).

Question 3 - Typing rules for checked exceptions Consider the following Java classes:

class E1 extends Exception { }
class E2 extends E1 { }
class E3 extends E2 { }
class Parent {
  void m () throws E2 { throw new E2(); }
  Parent f (Heir h) { return this; }
}
class Heir extends Parent {
  ...
}

Say, for each case below, what happens if we replace dots by the specified method declaration (please, at least 1-2 lines of explanation for each case; in case of static error explain what the error is).
1. void m () throws E3 { throw new E2(); }
2. void m () throws E1 { throw new E2(); }
3. void m () throws E3 { super.m(); }
4. void m () {
   try { super.m(); } catch (E1 e) { }
}
5. void m () throws E2 { super.m(); }
6. Heir f (Heir h) { return this; }
7. Parent f (Parent p) { return super.f(p); }
8. Parent f (Heir h) { return super.f(h); }

**Question 4 (optional extra-credit) - Application of reduction and typing rules for MINIJAVA with inheritance**

Let \( P \) be the following MINIJAVA (with inheritance) program

```java
class A {
    int f1;
    int m () {this.f1 = 5;}
}
class B extends A {
    int f2;
    int m () {this.f1 = 3; this.f2 = 7;}
}
```

1. Show the class type environment \( \Gamma_P \) which can be extracted from \( P \).
2. Prove, by applying the typing rules, that the following expression \( e \)

```
new A.m(); new B.m();
```

is well-formed in the class type environment \( \Gamma_P \) and the empty local environment.
3. Evaluate \( e \) starting from the empty store.

**Question 5 (optional extra-credit) - Implementation of recursive data structures in Java**

Assume that we have a Java class which implements forests (sequences of trees with integer labels), as shown below. The methods first and rest are assumed to be called only on a non-empty forest, and return the first tree and a forest equal to the receiver without the first tree, respectively. The method add returns a forest obtained adding as last tree element the argument.

```java
class Forest {
    Forest() { ... } // empty forest
    boolean isEmpty() { ... }
    Tree first () { ... }
    Forest rest () { ... }
    Forest add (Tree t) { ... }
}
```

1. Fill in the following definition of a class which implements trees with integer labels (a tree consists of an integer label and of the sequence of the children, which is a forest).

```java
class Tree {
    int label;
    Forest f;
    Tree (int label, Forest f) { this.label = label; this.f = f; }
    Tree (int label) { ... } //constructs a leaf
    int depth () { ... } // depth of the tree (maximal length of a path)
}
```

2. Add in Tree a method Tree pos() which returns a tree equal to the receiver but where all negative labels have been replaced by zero.