CMSC 245: Principles of Programming Languages  
Lab #7: Java Types

1. Without just downloading the code from this course, write a Java program that throws an ArrayStoreException.

2. *Heap pollution* can occur in a Java program when you omit the type variables to a polymorphic (generic) type. In JLS4.10.2, we see that, for a polymorphic class $C<A>$, the raw type (the one without type arguments) $C$ is a supertype of an instantiation $C<\tau>$. In JLS5.2, we see that we can perform an *unchecked conversion* when assigning one variable to another, and JLS5.1.9 describes an unchecked conversion as converting from a raw type $C$ to an instantiated one $C<\tau>$. This means that, by going via the raw type $C$, we can convert from a type $C<\tau_1>$ to $C<\tau_2>$, even if $\tau_1$ and $\tau_2$ are unrelated. When trouble strikes by these shenanigans, Java throws a ClassCastException, even where there isn't a cast.

Write a program that stores a String in an ArrayList<Integer> by using the raw type ArrayList to cause heap pollution. Then try calling intValue() on the String. You should get a ClassCastException.

3. Given the following methods, fill in the blanks so that the method is as general as possible – that is, can be called with the greatest number of different types of arguments. Types to consider include java.lang.Comparable, java.util.Collection, java.util.List, java.util.ArrayList, and java.util.Set.

   a. public static void addAll(...) to, ... from
   
   ```java
   { for(int i = 0; i < from.size(); i++)
     { to.add(from.get(i));
     }
   }
   ```

   b. public static ... findSmaller(...) where, ... what
   
   ```java
   { for(int i = 0; i < where.size(); i++)
     { if(where.get(i).compareTo(what) < 0)
       { return where.get(i);
       }
     }
   return null;
   ```