INTRODUCTION TO HASKELL—TYPE CLASSES

Your task: Peruse the following information about Haskell types,¹ and on the reverse of this page, draw a class hierarchy diagram.

Classes in **bold-face monospaced font** should be in your diagram. Items shown in *italic monospaced* are primitive types rather than typeclasses, and should be labeled within a typeclass category. Do the typeclasses first.

Haskell provides a rich collection of numeric types, based on those of Scheme, which in turn are based on Common Lisp. **Num** is a subclass of **Eq**, but not of **Ord**; this is because the order predicates do not apply to complex numbers. The subclass **Real** of **Num**, however, is a subclass of **Ord** as well.

The **Num** class provides several basic operations common to all numeric types; these include, among others, addition, subtraction, negation, multiplication, and absolute value. **Num** does not provide a division operator; two different kinds of division operators are provided in two non-overlapping subclasses of **Num**.

The class **Integral** provides whole-number division and remainder operations. The standard instances of **Integral** are **Integer** (unbounded or mathematical integers, also known as “bignums”) and **Int** (bounded, machine integers, with a range equivalent to at least 29-bit signed binary). A particular Haskell implementation might provide other integral types in addition to these. Note that **Integral** is a subclass of **Real**, rather than of **Num** directly.

All other numeric types fall in the class **Fractional**, which provides the ordinary division operator (/). The further subclass **Floating** contains trigonometric, logarithmic, and exponential functions. The **RealFrac** subclass of **Fractional** and **Real** provides a function properFraction.

Of the standard numeric types, **Int**, **Integer**, **Float**, and **Double** are primitive. The others are made from these by type constructors. **Complex** (found in the library Complex) is a type constructor that makes a complex type in class **Floating** from a **RealFloat** type.

**Eq** is used for types that support equality testing. **Ord** is for types that have an ordering. Members of **Show** can be presented as strings. **Read** is sort of the opposite typeclass of **Show**. The read function takes a string and returns a type which is a member of **Read**.

**Enum** members are sequentially ordered types — they can be enumerated. The main advantage of the **Enum** typeclass is that we can use its types in list ranges. They also have defined successors and predecessors, which you can get with the succ and pred functions. Types in this class: (), **Bool**, **Char**, **Ordering**, **Int**, **Integer**, **Float** and **Double**.

**Bounded** members have an upper and a lower bound.

Draw your diagram here. Don’t worry about getting it perfect; the idea is to explore the relationships!