Notes lecture 3 – Generics

* Different than C++ templates:
  + It does **not generate a new class** for each parameterized type
  + The **constraints can be imposed on the type variables** of the parameterized types.
* **Autoboxing**: automatic conversion between a primitive type and an object instance of a corresponding reference type
* **Generic methods**
  + *Static methods* cannot use the *type variables* of the class
    - **class BlaBla<T>** 🡺 static methods cannot use T
  + A generic method can contain **type variables different** than those used by **the generic class**
    - **class BlaBla<T>**
    - **<T,S> void altBlaBla(T sth, S sthElse)**
  + A **generic method** can be defined **in a non-generic class**
    - **class BlaBla**
    - **<T> void altBlaBla(T sth)**
* **Raw types** (should be avoided) are defined to be one of:
  + The reference type that is formed by taking the name of a **generic type declaration without an accompanying type argument list**.
  + An **array** type whose element type is a raw type.
  + A non-static member type of a raw type R that is not inherited from superclass or superinterface of R.
* **Generic arrays:**
  + Creation: **T[] elem=(T[])new Object[dim];**
* **Bounds:**
  + General form:
    - **T extends [C &] I1 [& I2 &...& In]**
    - T inherits the class C and implements the interfaces I1, ... In.
* **Wildcards:**
  + We use ? to denote any type (or unknown type)
* **Collections:**
  + **Iterator** code structure:

A computer code with blue and black text

Description automatically generated

* + **Iterable<T>** - needed for the data structures the need to be parsed using a “for-each” type of for:

A black text on a white background

Description automatically generated

* + **Comparable and Comparators**
    - Comparable interface implemented by objects that need to be compared
      * **compareTo()** method
    - Comparator objects that compare 2 objects