J12 Generics

- Generics
- Boxing classes
- Restrictions on generics

Generics

- Java allows us to parameterise a class, interface or method with one or more types; such a class/interface/method is called *generic*.
- Generics allow checking type correctness at *compile-time*.
- We can declare generic methods (with parameters whose type is a parameter) in a non-generic class
  - For examples, see `java.util.Arrays`.
- Type parameters can only be instantiated with class types (subclass of `Object`).

Declaring generics

```
class Pair<T, U> {
    private T first;
    private U second;
    Pair(T first, U second) { ... }
    T getFirst() { return first; }
}
static <T> void shuffle(T[] array) { ... }
```

- Type parameters in angle brackets (“< >”).

Using generics

- Specify actual types (values of type parameters) in angle brackets.

```
Pair<String, String> strPair =
    new Pair<String, String>(“a”, “b”);
```

- Not necessary to specify type parameters everywhere, if compiler can guess from context (*type inference*).

```
Pair<String, String> strPair = new Pair(“a”,“b”);
```

Type parameters

- Type parameters can only be instantiated with class types (subclass of `Object`).
  - Primitive types can not be used to instantiate a generic class/method.
  - Workaround: use “boxing” types.
- The only assumption we can make about a type parameter `T` is that it is an object: i.e. it extends `Object`.
  - All we can do with a variable/value of a generic type is pass it around and call methods defined by `Object`.

Boxing classes

- For each primitive type, Java has a corresponding *boxed* class:

  `int` – `Integer`, `double` – `Double`, `char` – `Character`, `boolean` – `Boolean`, and so on.
- These classes *wrap* a primitive value in an object
- Often used to pass primitive values into generic classes/methods.
- Java supports boxing/unboxing (i.e., wrapping/unwrapping) primitive values automatically ("autoboxing"):
  - `Integer i = 5;` (Boxing an `int` literal)
  - `int j = i;` (Unboxing to an `int` variable)

Boxing classes

- The Java boxing classes also collect (mostly static) methods and constants related to primitive types, for example:
  - `Integer.parseInt(String s)`
  - `Integer.MIN_VALUE`, `Integer.MAX_VALUE`
  - `Charater.isLetter(char c)`
- They are subclasses of `Object`, and can be subclassed.
- Boxing classes have a memory overhead compared to primitives, since they are instantiated as true objects.

Restricting type parameters

- *Bounds* can be placed on type parameters to make them “less generic”
  - For example,

    ```
    <T extends Comparable<T>> void sort(T[] a) {…}
    ```

  - This **restricts** the class types that can be used with the generic.
  - This **increases** the assumptions that can be made about a value of the generic type.
  - Note: Bounds can be classes or interfaces.

Type erasure

- The Java compiler uses type parameter values to check type correctness at compile time, but then removes this information from the compiled code (replacing the type parameter with `Object`, or its bounding type).
- This implies some limitations on generics:
  - Cannot have overloaded methods in a class that differ only in the instantiation of a type parameter.
  - Cannot create (`new`) instances of type parameters.
    - Workaround to create `T[]`.
  - Cannot have parametric type static fields.

https://docs.oracle.com/javase/tutorial/java/generics/restrictions.html