Restrictions on generics

1. Primitive types cannot be used generically. Pair<int, char> doesn't work. That's why Java has the wrapper classes and does automatic boxing an unboxing.

2. A generic class cannot extend class Throwable. The following two declarations are syntactically incorrect.

```
class C<T>  extends Throwable { … }
class C<T>  extends Exception { … }
```

3. A catch clause cannot catch an instance of a Type parameter. The following is syntactically incorrect:

```
public static <T extends Exception> void m(…) {
   try { … }
   catch (T e) { … }
}
```

4. An instance of a type parameter cannot be created. The code to the right is syntactically incorrect. This is because of type erasure. Before compiling, all type parameters are removed, so how can an instance of E be created? There are work-arounds for this, which we don't go into.

```
/** Create a new E and append it to b. */
public static <E> void append(ArrayList<E> b) {
   E e= new E();  // compile-time error
   b.add(e);
}
```

5. Arrays of parameterized types may not be created. The following expression is syntactically incorrect and won't be compiled.

```
new ArrayList<Integer>[2];
```

You can write a statement as shown below, thus creating a raw ArrayList. You will get an "unchecked conversion" warning, which you can ignore, because that raw ArrayList is stored in a1, which is of type ArrayList<Integer>[]. That's OK.

```
ArrayList<Integer>[] al= new ArrayList[2];
```

6. Static fields of type parameters are not allowed. Consider the class to the right and these two declaration/assignments:

```
Watch<Apple> iwatch= new Watch<>();
Watch<Samsung> swatch= new Watch<>();
```

```
public class Watch<T> {
   private static T os;
   …
}
```

Static field os is shared by iwatch and swatch. What is its type? It can't be both Apple and Samsung. Therefore, static fields of type parameters are not allowed.

7. The expression ob instanceof T is not allowed if T is a parameterized type. For example,

```
ob instanceOf Array<Integer>
```

is syntactically incorrect. This is because of type erasure. Parameterized types are removed before a syntactically correct program is compiled.

8. A method name cannot be overloaded if type erasure leads to the same raw type. Consider the two procedures declared below. Erase the parameterized type and they have the same signature. Therefore, this overloading is not allowed.

```
public void print(ArrayList<Integer> p) { … }
public void print(ArrayList<Double> p) { … }
```