Type Erasure

Generics concept is introduced in Java language to provide tighter type checks at compile time and to support generic programming. The way to implement generics, the Java compiler applies **type erasure** to:

* Replace all type parameters in generic types with their bounds or Object if the type parameters are unbounded. The produced bytecode, therefore, contains only ordinary classes, interfaces, and methods.
* Insert type casts if necessary to preserve type safety.
* Generate bridge methods to preserve polymorphism in extended generic types.

In general, the compiled generic code actually just uses java.lang.Object wherever you talk about T (or some other type parameter) – and there’s some metadata to tell the compiler that it really is a generic type. When you compile some code against a generic type or method, the compiler works out what you really mean (i.e. what the type argument for T is) and verifies at compile time that you’re doing the right thing, but the emitted code again just talks in terms of java.lang.Object – the compiler generates extra casts where necessary. At execution time, a List and a List are exactly the same; the extra type information has been erased by the compiler.