



Generics with Discretion and the Command Line

Corky Cartwright
Department of Computer Science
Rice University



How Generics Work

- The implementation of generics is smoke. The Java compiler (javac) understands the typing rules which are based on the idea that each instantiation of a generic class (*e.g.*, `List<Integer>`, `List<Number>`, `List<Object>`) is a disjoint type and enforces the appropriate typing rules based on this conceptual model. But the Java compiler implements generic types by erasing the parametric type information and automatically inserting casts where necessary. In a type-correct program (one that does not violate any typing rule and hence generates no errors or warnings), these casts can never fail.
- Note that subtyping is *not* co-variant: `A <: B` (is a subtype of) $\not\Rightarrow$ `List<A> <: List`. Co-variant subtyping does not hold because of object mutation. If I can change the elements embedded in a composite object consistent with the declared type of the composite object, I cannot soundly treat a `List<Integer>` as a `List<Object>`. These restrictions on generics are determined by the mathematics of type parameterization not the vagaries of Java.



Aside: Accommodating Co-variance

- Co-variant subtyping is mathematically feasible but it requires much more stringent type-checking rules, which make it unacceptable in most situations. Some language like Scala allow the programmer the option of specifying that a particular generic type is co-variant. Java uses a different approach called wildcards to accommodating occasional co-variance.
- The theory of wildcard types is nearly non-existent. The developers of Java 5.0 decided wildcards were so useful that they should be included in Java despite the fact that the research literature (at the time of Java 5.0's beta release) only included one fragmentary paper (published in a workshop) that discussed wildcard types (and a few others that talked about similar but technically different type systems). Unfortunately, the design in Java 5.0/6.0 is flawed. My recommendation: use only simple forms of wildcards and use them only when there is no other equally good way to produce type-correct code.



Raw Types

- The Java type system refers to the erased form of generic types as *raw* types. Raw types can be used in program text when integrating generic and non-generic code and when working around restrictions in generic type system imposed by erasure, but their usage should be minimized. Most usages of raw types constitute a breach of the Java generic type system. A program that uses raw types is likely to generate type warnings.
- No further discussion of raw types in this course.



Recommendations on Using Generics

- Ideally suited for parameterizing the types of classes that represent program data objects (including closures).
- Do not hesitate to use generics for clarifying the typing of data classes (and hence avoiding explicit casts).
- If a program generates a plethora of warnings, something is wrong. In general warnings should be avoided; they correspond to breaches in the type system and very likely correspond to semantic (run-time) type errors. Every warning message should be justified by a program comment.
- Remember that generics are supposed to serve the programmer, not vice versa



For Next Class

- Homework due on Friday. It consists of doing HW6 in Java given a Scheme solution.
- Language Levels bugs. External (file-crossing) references still are not resolved in some cases. The Language Levels facility is not designed to handle intermixed compilation involving `.java` files (`.dj2` files can be used instead). You can work around this bug by putting your program (except for the test class) in a single class, including the letters `Test` in your test file name, and separately compiling each file starting with you main program file. If you use separate file-by-file compilation, you can formulate your test file and perhaps your support files as `.java` files.
- Big tests (like bigData) will probably require enlarging the stack of the DrJava interactions JVM. Insert the argument string `-Xss64M` in the dialog box labeled JVM args for Interactions JVM in the Miscellaneous panel of DrJava Preferences.