Generic concept came at 1.5.

Generics basically used for type safety and to avoid type casting problems.

Exercise1: Show arrays type safety

String str[] = new String[2];

str[0]=”sudheer”;

str[1]=10; // compile time err saying Incompatible types.

Note: Arrays are by default type safety, String array can only contain strings, Integer array can only contain integers, etc.

Excercise2: show problems of non-generic code with java.

List list = new ArrayList (); //suppose to contain student names, by mistake if it contain other data like number, what happens?

list.add(“sudheer”);

list.add(10);

Sring name1= (String)list.get(0);

Sring name2= (String)list.get(1); // RunTime Error: java.lang.ClassCastException: java.lang.Integer cannot be cast to java.lang.String

Solution: List<String> list = new ArrayList<String>();

list.add(“sudheer”);

list.add(10); //Compile Time error saying, Integer not applicable in place of String.

Code that uses generics has many benefits over non-generic code:

* Stronger type checks at compile time.  
  A Java compiler applies strong type checking to generic code and issues errors if the code violates type safety. Fixing compile-time errors is easier than fixing runtime errors, which can be difficult to find.
* Elimination of casts.  
  The following code snippet without generics requires casting:
* List list = new ArrayList();
* list.add("hello");
* String s = **(String)** list.get(0);

When re-written to use generics, the code does not require casting:

List<String> list = new ArrayList<String>();

list.add("hello");

String s = list.get(0); // no cast

* Enabling programmers to implement generic algorithms.  
  By using generics, programmers can implement generic algorithms that work on collections of different types, can be customized, and are type safe and easier to read.