How to do it using Collections

In collections we can use ArrayList to store a list of Student objects like as:

1. **class Student**
2. **{**
3. **public int RollNo{get; set;}**
4. **public string Name{get; set;}**
5. **}**
7. ***//List of students***
8. **ArrayList studentList = new ArrayList();**
9. **Student objStudent = new Student();**
10. **objStudent.Name = "Rajat";**
11. **objStudent.RollNo = 1;**
13. **studentList.Add(objStudent);**
15. **objStudent = new Student();**
16. **objStudent.Name = "Sam";**
17. **objStudent.RollNo = 2;**
19. **studentList.Add(objStudent);**
21. **foreach (Object s in studentList)**
22. **{**
23. ***//Type-casting. If s is anything other than a student***
24. **Student currentStudent = (Student)s;**
25. **Console.WriteLine("Roll # " + currentStudent.RollNo + " " + currentStudent.Name);**
26. **}**

Problem with Collections

Suppose by mistake you have added a string value to ArrayList like as

1. **studentList.Add("Generics"); *//Fooling the compiler***

Since ArrayList is a loosely typed collection and it never ensure compile-time type checking. Hence above statement will compile without error but it will throw an InvalidCastException at run time when you try to cast it to Student Type.

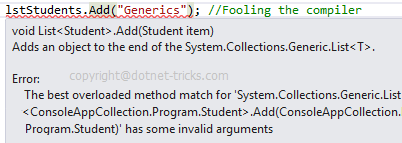
How to do it using Generics

In generics we can use generic List to store a list of Student objects like as:

1. **List<Student> lstStudents = new List<Student>();**
3. **Student objStudent = new Student();**
4. **objStudent.Name = "Rajat";**
5. **objStudent.RollNo = 1;**
7. **lstStudents.Add(objStudent);**
9. **objStudent = new Student();**
10. **objStudent.Name = "Sam";**
11. **objStudent.RollNo = 2;**
13. **lstStudents.Add(objStudent);**
15. ***//Looping through the list of students***
16. **foreach (Student currentSt in lstStudents)**
17. **{**
18. ***//no need to type cast since compiler already knows that everything inside***
19. ***//this list is a Student***
20. **Console.WriteLine("Roll # " + currentSt.RollNo + " " + currentSt.Name);**
21. **}**

Advantage with Generics

In case of collections you can make fool to compiler but in case of generics you can't make fool to compiler as shown below. Hence Generics provide Type Safety.



Cleaner Code with Generics

Since compiler enforces type safety with Generics. Hence fetching data from Generics doesn't required type casting which means your code is clean and easier to write and maintain.

Better Performance with Generics

As you have seen in above example, at the time of fetching the data from the ArrayList collection we need to do type casting which cause performance degrades. But at the time of fetching the data from the generic List we don't required to do type casting. In this way Generics provide better performance than collections.