

AITP NCC 2002 Annual Java Competition Event

AITP NCC 2002 Java Competition Event - Kansas City, Missouri

1 st Place	Team 16	248
2 nd Place	Team 12	171
3 rd Place	Team 10	138
4 th Ranking	Team 08	125
5 th Ranking	Team 14	106

Team Notes:

Team 16: 1st Place

Better approach would be to reuse the frames and panels by setting visibility. Database revision and write not functional.

Team 12: 2nd Place

Architectural approach was on right track. Poor documentation, especially in core areas. Earnings is a computed field and should not appear on the GUI as an input field.

Team 10: 3rd Place

Use of protected attributes is evil - use private instead. Do not comment out code - delete it if you are not going to use it. You did not use the package statement.

Team 08: 4th Ranking

A monolithic class structure is evil - you need to take a class collaboration approach - this is one of the key advantages to an object-oriented language such as Java (this also prevented you from picking up points for implementation and problem solving approach and object-oriented architectural characteristics - 40% of the possible quantitative score). Documentation was very good - you picked up the maximum number of points here.

Team 14: 5th Ranking

SQL import not needed (you did not use it - and if you had it would have been a show stopper since the dm package provided all of the required SQL services). Your documentation was very good.

All Teams:

Remember, we did not "subtract" points from any of the teams. The only time we would do this was outlined in the contest rules and in the pre-contest briefing - none of the teams that participated this year lost points.

AITP NCC 2002 Annual Java Competition Event

Judges' Comments:

Each of the five teams above were building the required functionality; however, the architectural approaches were very different. Team 08 built a monolithic graphical user interface class that attempted to provide all of the menu services inside one class. This approach presented the team with two problems: First, since this is not an object-oriented approach, their solution did not meet the requirements to pick up any of the available 40% possible points relating to architecture. Second, this approach is more difficult to debug since there is a lot more code to keep track of in the same class. Team 10 took an interesting approach by using protected attributes. As a general rule, this approach is very weak since it allows the attributes to be accessed and altered by other classes in the same package without going through the class method services designed to support the attributes. This close-coupled approach should be avoided unless there is a very strong justification for not using private attributes, which is typically rare. Team 16 was clearly on the right track.

Several patterns were more evident at this year's competition event compared with the past two events:

- Many of the teams were relying only on a single textbook - many of these textbooks were very weak and are really not a good choice for higher education institutions. I will return to this in a moment. Using reference material from at least two different authors is always a good idea since you will be exposed to different techniques that can be used to solve the same problem - this enhances the learning experience, which leads to better problem solving skills.
- Many of the teams were trying to build monolithic classes. This observation coupled with the poor choice of textbooks above leads the judges to the conclusion that many of the teams were probably being taught Java by faculty with little to no Java (or other object-oriented programming language) experience. The use of a monolithic class (one class is used to solve the problem instead of several classes working together) is an approach that would typically be taken with a structured language such as COBOL. Many instructors will choose simpler textbooks because they need to teach themselves the material as they are teaching the students. Java is first and foremost an object-oriented design programming language. The fundamental paradigm shift required from structured programming to object-oriented programming requires a different approach to learning on the part of the instructor. The instructor must understand the fundamental concepts behind class collaboration design.

AITP NCC 2002 Annual Java Competition Event

- During the competition event briefing, only half of the teams indicated that they have had exposure to UML (the Unified Modeling Language). Java and UML go hand-in-hand. Object-Oriented design concepts and UML need to be taught before students learn Java - otherwise the students are losing too much in the learning process. Experience indicates that twice as much material can be covered when Java is taught using UML since students learn the concepts much faster through the visualization process. To this end, UML tutorials have been prepared and placed on the AITP NCC Java Competition Website to teach the UML concepts that are required for the contest.

Some interesting observations on team problem solving approaches were made:

- Essential Use Cases basically describe how a user interacts with the system through a graphical user interface (GUI), which is a technical term for a menu. It should have been obvious that each Essential Use Case was basically a description of a particular menu; therefore, at least one class should be built for each essential Use Case. From a design perspective, common functionality and attributes would be prime candidates for inclusion in abstract classes.
- By examining the UML documentation, it should have been obvious that there was common functionality that was used by both the Boss and HourlyWorker GUI menus. This common functionality was shared through an abstract class in both the hr and dm package classes - this was an obvious approach for the gui package classes.
- Even if an abstract class were not used to factor out common attributes and functionality, there was enough similarity to justify building the Boss classes first (before the HourlyWorker classes). This approach would save time since much of the Boss class code could be copied into the HourlyWorker classes where the required changes could be made. Most of the teams tried to develop both the Boss and HourlyWorker classes in parallel, which wasted time (it is hard to maintain focus when you are attempting two separate tasks at the same time). The Boss classes are simpler to implement, which is why they are the best candidates for building first.

AITP NCC 2002 Annual Java Competition Event

- Many of the teams had difficulty trying to figure out where to start - one hint was in the test package classes. At the start of the contest, we had each team run the test classes to make sure that the java development environment was configured correctly on each computer. Since the test package classes exercised the same methods that the gui package classes would be required to use, each team should have taken a look at how the test package classes were constructed (especially since all of the source code and UML diagrams were provided to each team).

Logistical issues that deserve comment:

- The AITP NCC 2002 Java Competition Event consisted of two parts: A pre-contest briefing scheduled before lunch and the actual contest event scheduled after lunch. This year, more contests were held at the same time compared to prior years. Consequently, a few teams were unable to attend the pre-contest briefing. As a work around, all teams that had a schedule conflict were provided an opportunity to be briefed individually. Additionally, security issues associated with the Vice President's visit created a delay in setting up the Gateway computer lab, which required some preparation work by the teams in order to get the contest environment properly configured and setup. Despite these issues, feedback from the teams indicated that the participants felt that the contest event was fair and that they were given adequate support for the contest event.
- The fact that this year's problem statement was based on last year's problem statement and solution presented an interesting opportunity for any of the teams that actually practiced trying to solve last year's problem statement - the same skills were required for this year's contest (we assume that students know how to build GUI's since this is one of the first things taught in class). Interestingly, many of the teams approached their solution by using the books that they had (it is a bit amusing to be able to identify an author and book by the code in the submitted solution - so at least the judges were entertained a bit). Although mentioned previously in this document, it should be emphasized that students need to rely on more than one source of reference when learning a programming language - otherwise it can be difficult to approach a problem. This also points out a need to practice on previous year's problem statements - in this case, there would have been a significant advantage since last year's problem statement was reused as a foundation for this year's contest.

AITP NCC 2002 Annual Java Competition Event

- The post-contest briefing was well attended and the contestants, faculty, and other audience members were provided time to discuss the contest problem statement and ask questions. Java curriculum issues were also addressed.

There is an important paradigm shift required when moving from a structured programming language like COBOL to an object-oriented programming language like Java. The ability to build specialists classes to perform specific functions and then build collaborations of classes to deliver services is one of the most important benefits. Class collaborations allow common functionality to be factored out, faster class construction time - since smaller classes are easier to debug and test, and easier to add additional functionality to since future services can be added through new modules (in Java, the package is a good technique). Each year, the Java problem statement is designed to assist educators teach Java by providing a miniature Java case study that can be dealt with in a single semester class.

When approaching a programming problem, keep in mind that there are always three basic elements:

- The set of services and associated information that needs to be processed
- A mechanism for interfacing with the system in order to utilize the services
- A persistence mechanism for retaining the information on a long-term basis

The specific Java skills required to complete a given problem statement are covered in the rules portion of this web site. Essentially, a team needs to have skills in the following:

- JDBC-ODBC
- SQL
- Streams
- JavaDoc
- How to import specific classes vs. global importing using `.*`
- How to use getter (assessor) and setter (mutator) methods
- Use of the `toString` method
- String class
- Wrapper classes
- AWT and layout knowledge
- Applets
- Applications
- Listeners
- Basic Java API's
- Exception handling (try-catch)

AITP NCC 2002 Annual Java Competition Event

Specifically, each problem statement will involve a relational database and a class collaboration problem solving approach.

Comments on the scoring system:

The maximum points that a team could score on the 2002 AITP NCC Java Competition Event was 450 points as follows:

Points	%	Category
210	50%	Satisfying Requirements
126	30%	Implementation and Problem Solving Approach
42	10%	Internal and JavaDoc Documentation
<u>42</u>	<u>10%</u>	Object-Oriented Architecture Characteristics
420	100%	Total Possible Quantitative Score
<u>30</u>		Maximum Available Subjective Points
450		Maximum Possible Score

Each team had a realistic chance of picking up the basic quantitative points for a maximum score of 420. In order to pick up any of the subjective points ("Wow" points for impressing the judges), a team would have had to submit an impressive solution. None of the teams picked up any Wow points this year (One team picked up Wow points in the 2000 Java contest and one team picked up Wow points in the 2001 Java contest).

The scoring system is designed around the following benchmarks:

105 Points: An strong "B" or "A" student that has completed at least one semester of Java should be able to score 105 points or higher.

147 Points: A high "A" student that has completed at least one semester of Java should be able to score 147 points or higher.

To qualify for honorable mention we would expect a minimum score of 105 or higher.

210 Points: A strong team that has had at least two semesters of Java or that has experimented, practiced, or played around with Java should be able to score 210 points or higher. Breaking this point benchmark would require more sophisticated knowledge. The scoring system took into account the different possible solutions and this would be reflected at this level.

AITP NCC 2002 Annual Java Competition Event

252 Points: This is the upper range that we would expect from a student team that has only had limited exposure to Java in the classroom. Without more complicated Java work, a student would be challenged to score higher than this benchmark. Typically, students that score in this range are writing their own Java programs, perhaps running a web site, are active hobbyists, or have done some programming for someone else outside of the classroom.

294 Points: A team that reaches this level is professionally employable with a little training.

450 Points: A team scoring at this level probably already has a job programming Java, which would lead one to ask if they should really be participating in the Java competition event (since it is for students with less than one year professional experience).

It should be noted that the benchmark scores 105, 147, 210, and 252 are based strictly on the quantitative points and do not take into account the subjective Wow points. However, these benchmarks apply equally well for a team that picked up Wow points, since this represents a professional level of Java programming technique. It would be very unusual for a team to actually score the full 450 points.

A comment on why we chose such a wide range for the point scoring is in order. First, a programming language typically offers a number of ways to solve any given problem. With this multiple solving problem approach in mind, the AITP NCC Java Competition Event Problem Statement Committee develops a problem statement and then examines the various ways the problem statement can be solved. Next, we rank each approach based on the current literature related to Java programming. Textbook solutions that are typically taught in class are given a base score. Professional level solutions are given a doubled score. This reflects the higher-level knowledge and skill that is required to implement a more sophisticated solution. For example, An AWT approach is worth basic points, while a Swing approach is worth double points - because Swing is a more advanced technique.

The key to doing well in the Java contest is to work fast and pick up as many points as possible - making sure that your solution compiles and runs without crashing. This is easy to accomplish providing a team takes the common sense approach to work on one function point at a time on class at a time. Save frequently and use method stubs to speed up the build process.

One final remark, always make sure that you check out all of the information and directory contents. We provide a lot of information to assist the teams (after all, in the real world, you could expect to have a certain amount of support on a programming project - we have attempted to simulate this).