

Mathematics

AP Computer Science - Java

9/18/2009

Object-Oriented Program Design	Program Implementation	Program Analysis
<p>The overall goal for designing a piece of software (a computer program) is to correctly solve the given problem. At the same time, this goal should encompass specifying and designing a program that is understandable, can be adapted to changing circumstances, and has the potential to be reused in whole or in part. The design process need is to be based on a thorough understanding of the problem to be solved.</p>	<p>The overall goals of program implementation parallel those of program design. Classes that fill common needs should be built so that they can be reused easily in other programs. Object-oriented design is an important part of program implementation.</p>	<p>The analysis of programs includes examining and testing programs to determine whether they correctly meet their specifications. It also includes the analysis of programs or algorithms in order to understand their time and space requirements when applied to different data sets.</p>
<ol style="list-style-type: none"> 1. Program design <ol style="list-style-type: none"> A. Read and understand a problem description. B. Apply data abstraction and encapsulation. C. Read and understand class specifications and relationships among the classes (“is-a”, “has-a” relationships). D. Understand and implement a given class hierarchy. E. Identify reusable components from existing code using classes and class libraries. 2. Class design <ol style="list-style-type: none"> A. Design and implement a class. B. Design an interface. C. Choose appropriate data representation and algorithms. D. Apply functional decomposition. E. Extend a given class using inheritance. 	<ol style="list-style-type: none"> 3. Implementation techniques <ol style="list-style-type: none"> A. Methodology <ol style="list-style-type: none"> 1. Object-oriented development 2. Top-down development 3. Encapsulation and information hiding 4. Programming constructs <ol style="list-style-type: none"> A. Primitive types vs. objects B. Declaration <ol style="list-style-type: none"> 1. Constant declarations 2. Variable declarations 3. Class declarations 4. Interface declarations 5. Method declarations 6. Parameter declarations C. Console output (system.out.print/printin) D. Control <ol style="list-style-type: none"> 1. Methods 2. Sequential 3. Conditional 4. Iteration 5. Recursion 5. Java library classes (included in the A level AP Java Subset) 	<ol style="list-style-type: none"> 6. Testing <ol style="list-style-type: none"> A. Test classes and libraries in isolation. B. Identify boundary cases and generate appropriate test data C. Perform integration testing 7. Debugging <ol style="list-style-type: none"> A. Categorize errors: compile-time, run-time, logic B. Identify and correct errors C. Techniques: use a debugger, add extra output statements, hand-trace code 8. Understand and modify existing code 9. Extend existing code using inheritance 10. Understand error handling <ol style="list-style-type: none"> A. Understand runtime exceptions 11. Reason about programs <ol style="list-style-type: none"> A. Pre-and post-conditions B. Assertions 12. Analysis of algorithms <ol style="list-style-type: none"> A. Informal comparisons of running times B. Exact calculation of statement execution counts 13. Numerical representations and limits <ol style="list-style-type: none"> A. Representations of numbers in different bases B. Limitations of finite representations (e.g., integer bounds, imprecision of floating-point representations, and round-off error)

Mathematics

AP Computer Science - Java

9/18/2009

Object-Oriented Program Design	Program Implementation	Program Analysis
<p>The overall goal for designing a piece of software (a computer program) is to correctly solve the given problem. At the same time, this goal should encompass specifying and designing a program that is understandable, can be adapted to changing circumstances, and has the potential to be reused in whole or in part. The design process need is to be based on a thorough understanding of the problem to be solved.</p>	<p>The overall goals of program implementation parallel those of program design. Classes that fill common needs should be built so that they can be reused easily in other programs. Object-oriented design is an important part of program implementation.</p>	<p>The analysis of programs includes examining and testing programs to determine whether they correctly meet their specifications. It also includes the analysis of programs or algorithms in order to understand their time and space requirements when applied to different data sets.</p>
<ol style="list-style-type: none"> 1. Program design <ol style="list-style-type: none"> A. Read and understand a problem description. B. Apply data abstraction and encapsulation. C. Read and understand class specifications and relationships among the classes (“is-a”, “has-a” relationships). D. Understand and implement a given class hierarchy. E. Identify reusable components from existing code using classes and class libraries. 2. Class design <ol style="list-style-type: none"> A. Design and implement a class. B. Design an interface. C. Choose appropriate data representation and algorithms. D. Apply functional decomposition. E. Extend a given class using inheritance. 	<ol style="list-style-type: none"> 3. Implementation techniques <ol style="list-style-type: none"> A. Methodology <ol style="list-style-type: none"> 1. Object-oriented development 2. Top-down development 3. Encapsulation and information hiding 4. Programming constructs <ol style="list-style-type: none"> A. Primitive types vs. objects B. Declaration <ol style="list-style-type: none"> 1. Constant declarations 2. Variable declarations 3. Class declarations 4. Interface declarations 5. Method declarations 6. Parameter declarations C. Console output (system.out.print/printin) D. Control <ol style="list-style-type: none"> 1. Methods 2. Sequential 3. Conditional 4. Iteration 5. Recursion 5. Java library classes (included in the A level AP Java Subset) 	<ol style="list-style-type: none"> 6. Testing <ol style="list-style-type: none"> A. Test classes and libraries in isolation. B. Identify boundary cases and generate appropriate test data C. Perform integration testing 7. Debugging <ol style="list-style-type: none"> A. Categorize errors: compile-time, run-time, logic B. Identify and correct errors C. Techniques: use a debugger, add extra output statements, hand-trace code 8. Understand and modify existing code 9. Extend existing code using inheritance 10. Understand error handling <ol style="list-style-type: none"> A. Understand runtime exceptions 11. Reason about programs <ol style="list-style-type: none"> A. Pre-and post-conditions B. Assertions 12. Analysis of algorithms <ol style="list-style-type: none"> A. Informal comparisons of running times B. Exact calculation of statement execution counts 13. Numerical representations and limits <ol style="list-style-type: none"> A. Representations of numbers in different bases B. Limitations of finite representations (e.g., integer bounds, imprecision of floating-point representations, and round-off error)