A4 - Due March 28 2016

Goal

You will modify your bookstore management program from A3 to implement a templated collection class. You will also incorporate polymorphism and abstract classes into your program.

Objectives

- design and implement classes that use polymorphism
- develop a templated class

Instructions:

1. Templating the Course collection

You will modify your course collection class (**`CourseArray`**), and make it a templated class. Your class must provide all the same member functions as in A3, including the overloaded operators, but it must be templated to contain **any type** of elements, including potentially non-pointer elements. If you are using the CourseArray class, you should rename it to Array.

2. Incorporating polymorphism

You will come up with a **new feature** for the bookstore management system that implements **polymorphism**. Your new feature must:

- provide the user with a new major set of functionality that does not already exist in the program
  - for example, "add course" and "delete course" are all features that already exist
- allow the user to select this new feature from the main menu in order to launch it
- be non-trivial, i.e. it must add significant code to your program
  - for example, adding a function in each class that returns the type of class is *not a feature*, and it is *trivial*
- involve at least one abstract class and at least four concrete classes that make sense in the real world
- have at least one function that is invoked polymorphically
  - do not implement polymorphism in constructors; object creation is almost never polymorphic unless the Factory design pattern is used
- be documented in your readme file; explain what the feature does, and which classes (abstract and concrete) and which functions (virtual or not) are involved in implementing the polymorphic behaviour
- be represented in the program UML diagram

Constraints

- your program must follow the existing design of the base code, including the encapsulation of control, UI, entity and array object functionality
- do not use any classes or containers from the C++ standard template library (STL)
- do **not** use any global variables or any global functions other than **`main`**
- do **not** use structs; use classes instead
- objects must always be passed by reference, not by value

Submission

You will submit:

- a UML class diagram that corresponds to the program design
- one **`tar`** file that includes:
  - all source and header files for your program
  - a Makefile

Grading

Marking components:

- Templated collection class: 30%
  5 marks: Correct definition of data members
  5 marks: Correct implementation of += that takes a single object
  5 marks: Correct implementation of += that takes a collection
  5 marks: Correct implementation of -= that takes a single object
  5 marks: Correct implementation of -= that takes a collection
  5 marks: Correct definition of templated collection in owner class
- UML diagram: 10%
  10 marks: presence of 1 abstract class and 4 concrete classes that make sense (2 marks each), with correct associations
- Polymorphism: 60%
  The marking for this feature is rubric based. First a number of points out of 4 is assigned, based on the table below. Then the number of points is multiplied by 15, which yields a grade out of 60.

0 points (Inadequate)	**2 points (Adequate)**	**4 points (Excellent)**
• no polymorphism is implemented • code uses RTTI or dynamic cast to check object type • new feature is trivial • new feature is not accessible from menu	• some polymorphism is implemented • all abstract and concrete classes are present • new feature has substance • new feature is accessible from menu	• polymorphism is fully implemented • all abstract and concrete classes are present • new feature is substantial and creative • new feature is accessible from menu

Deductions:

- Packaging errors:
  - 10 marks for missing Makefile
  - 10 marks for **consistent** failure to correctly separate code into source and header files
  - 10 marks for bad style
- Major programming and design errors:
  - 50% of a marking component that uses global variables, global functions, or structs
  - 50% of a marking component that consistently fails to use correct design principles
  - 50% of a marking component that uses prohibited library classes or functions
  - 50% of a marking component where unauthorized changes have been made to provided code
- Execution errors:
  - 100% of a marking component that cannot be tested because the code does not compile or execute
  - 100% of a marking component that cannot be tested because the feature is not used in the code
  - 100% of a marking component that cannot be tested because data cannot be printed to the screen