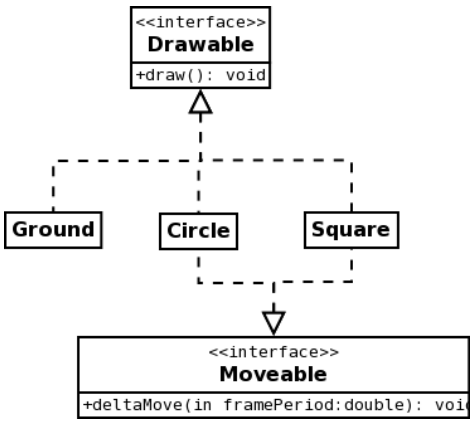
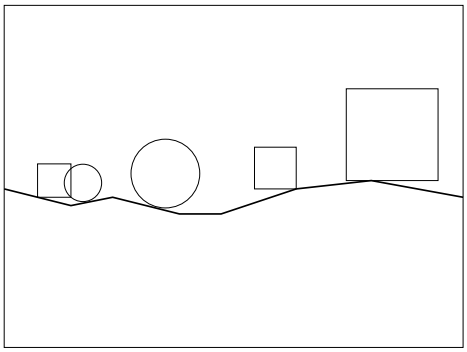


(a) A required subset of classes



(b) A possible scene

Figure 1: Illustrations

BRIEF INSTRUCTIONS:

- **Mon: March 9, 2015 @ 23:59**
- Submit your assignment on Moodle.
- Except for special circumstances, there are no extensions for the deadline. Please talk to your instructor if in doubt.
- Please develop the code individually. You are encouraged to collaborate with your colleagues at a conceptual level, in front of a sheet of paper. You may use UML when discussing problems, but do not share source code. Please consult the section on academic offenses in the calendar, starting at page 78.

Problem: [30 pts] Interfaces. You are asked to implement a simple figure animation program in 1.5 dimensions (in the style of Donkey Kong computer game of the 80's: `http://en.wikipedia.org/wiki/Donkey_Kong`) whose design is based on the concept of interface. Please implement in C++. The following are the functional and the design requirements:

a) When executed, the program will display a 1.5 dimensional scene (a vertical slice from a 3D world). The scene consists of ground and two kinds of figures: circles and squares. The set of figures is randomly generated. Figures move, with constant speed, on the ground, from the left side of the window to the right side (see Fig. 1b for an illustration of a possible scene). The window wraps around, so once figures reach the right side of the window, they will appear on the left side and they will continue to move until the program is terminated.

b) The ground is represented by an x-monotone polygonal chain (see `http://cgm.cs.mcgill.ca/~godfried/teaching/pr-projects/tallman/fig3.gif` for an illustration that clarifies this notion).

c) You are asked to create a design that includes the classes and relationships illustrated in the UML class diagram from Fig. 1a. Please use the method names suggested in the diagram.

d) Your starting point for the implementation is `/home/lib2720/assignments/1_shapes`. Please copy the contents of this directory in a working directory and then use the working directory. This project compiles a sample animation of a square class. Class *Simulator* hides the details of setting the Allegro library up for animation. Classes *SquareSimulator* and *EmptySimulator* illustrate the use of class *Simulator* to perform your own animations.

e) You must create single instances of the ground and figure objects. In your simulator class, you will need to draw figures and move figures. **You must draw and move via the interface**. You should use two collections of the appropriate interface pointers in your simulator class, one for the *Moveable* interface, and one for the *Drawable* interface. Use *shared_ptr* (C++11) to point to the actual figure objects.

f) Function *deltaMove* receives one parameter of type *double* representing the time period between two successive frames in the animation. Figures, when constructed, should be initialized with a value for speed. Speed coupled with argument *framePeriod* will allow the object to calculate its new position for the next animation frame.
The allegro documentation is available from `http://alleg.sourceforge.net/`

Marking guide:
- Comment header files (each class: the purpose for the class; each method: the purpose for the method, the parameters, return value, and any noteworthy observations).
- Choose how you assign responsibilities carefully: the main function should create the objects, but drawing and moving should be called from the class that has the information required for drawing (for example, the class that knows at what time intervals the scene needs to be updated).
- Use interfaces wherever possible, except, of course, when binding the concrete classes to the interfaces. Functions for draw and move should be called only via the interface. Use *shared_ptr* to refer to concrete figure objects.
- Avoid memory leaks.
- Use the 'const' keyword as much as possible and where appropriate.
- Avoid using methods that are excessively complex and long.
- Use names for objects and variables that are intuitive.
- Comment code that may be difficult to understand.
- Use a consistent scheme for naming classes, class members, functions, and variables.

SUBMISSION INSTRUCTIONS: Create a zip file containing the following:

- The source code for your project.
- The Makefile which is used to build the project.

Submit the zip file on Moodle before the deadline.