1. (5 points) Give the output for the following program.

```
#include <iostream>
#include <vector>
class Player {
public:
  Player() { std::cout << "default" << std::endl; }
  Player(int) { std::cout << "convert" << std::endl; }
  Player(const Player&) { std::cout << "copy" << std::endl; }
  Player& operator=(const Player&) {
    std::cout << "assign" << std::endl;
    return *this;
  }
private:
  int cp;
};
int main() {
  std::vector<Player> players;
  for (unsigned int i = 0; i < 2; ++i) {
    players.push_back( i );
  }
}
```

```
convert
copy
convert
copy
copy
```

---

2. (5 points) Give the output for the following program.

```
#include <iostream>
#include <vector>
class Player {
public:
  Player() { std::cout << "default" << std::endl; }
  Player(int) { std::cout << "convert" << std::endl; }
  Player(const Player&) { std::cout << "copy" << std::endl; }
  Player& operator=(const Player&) {
    std::cout << "assign" << std::endl;
    return *this;
  }
private:
  int cp;
};
int main() {
  std::vector<Player> players;
  players.reserve(2);
  for (unsigned int i = 0; i < 2; ++i) {
    players.push_back( i );
  }
}
```

```
convert
copy
convert
copy
```

3. (5 points) Give the output for the following program.

```
#include <iostream>
#include <vector>
class Player {
public:
  Player() { std::cout << "default" << std::endl; }
  Player(int) { std::cout << "convert" << std::endl; }
  Player(const Player&) { std::cout << "copy" << std::endl; }
  Player& operator=(const Player&) {
    std::cout << "assign" << std::endl;
    return *this;
  }
private:
  int cp;
};
int main() {
  std::vector<Player> players;
  players.reserve(2);
  for (unsigned int i = 0; i < 2; ++i) {
    players.emplace_back( i );
  }
}
```

```
convert
convert
```

---

4. (5 points) The following program will not compile. Why? What line number causes the problem?

```
#include <iostream>

class Shape {
public:
  Shape() {}
  virtual float area() const = 0;
};

int main() {
  Shape* shape = new Shape;
}
```

```
Line 6 or line 10 is the problem. You cannot instantiate an abstract base class
```

5. (10 points) Give the output for the following program.

```
#include <iostream>

class Shape {
public:
  Shape()  { std::cout << "Shape: default " << std::endl; }
  ~Shape() { std::cout << "Shape: destructor" << std::endl; }
};

class Circle : public Shape {
public:
  Circle()  { std::cout << "Circle: default " << std::endl; }
  ~Circle() { std::cout << "Circle: destructor" << std::endl; }
};

int main() {
  Shape* shape = new Circle;
}
```

```
Shape: default
Circle: default
```

---

6. (10 points) Give the output for the following program.

```
#include <iostream>

class Shape {
public:
  Shape()  { std::cout << "Shape: default " << std::endl; }
  ~Shape() { std::cout << "Shape: destructor" << std::endl; }
};

class Circle : public Shape {
public:
  Circle()  { std::cout << "Circle: default " << std::endl; }
  ~Circle() { std::cout << "Circle: destructor" << std::endl; }
};

int main() {
  Shape* shape = new Circle;
  delete shape;
}
```

```
Shape: default
Circle: default
Shape: destructor
```

7. (20 points) Write functions print and findTree, used on lines #9 and #13. print prints the *key* **and** *value* for each item in the map. findTree searches mymap for tree.

```
1  #include <iostream>
2  #include <vector>
3  #include <map>
4  #include <string>
5
6  void print(const std::map<std::string, int>& mymap) {
7    for (const auto& n : mymap) {
8      std::cout << n.first << ", " <<  n.second << std::endl;
9    }
10 }
11
12 bool findTree(const std::map<std::string, int>& mymap,
13                const std::string& name) {
14   return mymap.find(name) != mymap.end();
15 }
16
17 int main() {
18   std::map<std::string, int>
19     mymap = { {"Oak", 77}, {"Chestnut", 45}, {"Elm", 88} };
20   print(mymap);
21
22   std::string tree;
23   std::cin >> tree;
24   if ( findTree(mymap, tree) ) {
25     std::cout << mymap[tree] << " found" << std::endl;
26   }
27 }
```

8. (20 points) Write functions printRadius and removeRectangles, used on lines #40 and #41. printRadius prints the radius of the circles and removeRectangles removes all the rectangles in list shapes.

```
1  #include <iostream>
2  #include <list>
3  #include <string>
4  class Shape {
5  public:
6    Shape(const std::string& n) : name(n) {}
7    virtual ~Shape() {}
8    const std::string& getName() const { return name; }
9  private:
10   std::string name;
11 };
12 class Circle : public Shape {
13 public:
14   Circle(const std::string& n, float r) : Shape(n), radius(r) {}
15   float getRadius() const { return radius; }
16 private:
17   float radius;
18 };
19 class Rectangle : public Shape {
20 public:
21   Rectangle(const std::string& n) : Shape(n) {}
22 };
23 void printShapes(const std::list<Shape*>& shapes) {
24   for ( const Shape* const shape : shapes ) {
25     std::cout << shape->getName() << std::endl;
26   }
27 }
28
29 void cleanUp(std::list<Shape*>& shapes) {
30   for ( Shape* const shape : shapes ) {
31     delete shape;
32   }
33 }
34
35 void removeRectangles(std::list<Shape*>& shapes) {
36   auto it = shapes.begin();
37   while ( it != shapes.end() ) {
38     Rectangle* rect = dynamic_cast<Rectangle*>(*it);
39     if ( rect ) {
40       delete rect;
41       it = shapes.erase( it );
42     }
43     else ++it;
44   }
45 }
46
47 void printRadius1(const std::list<Shape*>& shapes) {
48   for ( Shape* const shape : shapes ) {
49     Circle* circle = dynamic_cast<Circle*>(shape);
50     if ( circle ) {
51       std::cout << circle->getRadius() << std::endl;
52     }
53   }
54 }
55
56 void printRadius2(const std::list<Shape*>& shapes) {
```

```
57    for ( Shape* const shape : shapes ) {
58      if ( dynamic_cast<Circle*>(shape) ) {
59        std::cout << static_cast<Circle*>(shape)->getRadius() << std::endl;
60      }
61    }
62  }
63
64  int main() {
65    std::list<Shape*> shapes;
66    const int n = rand()%25 + 5;
67    for ( int i = 0; i < n; ++i ) {
68      if ( rand()%2 ) {
69        shapes.push_back( new Circle("circle", rand()%25+5) );
70      }
71      else {
72      shapes.push_back( new Rectangle("Rectangle") );
73      }
74    }
75    printRadius1(shapes);
76    removeRectangles(shapes);
77    printShapes(shapes);
78    cleanUp(shapes);
79  }
```

9. (5 points) Give the output for the following program.

```
#include <iostream>
#include <cstring>

int main() {
  int x = 17;
  int y = 109;
  const int * q = &x;
  int& ref = x;
  ref = y;
  std::cout << x << std::endl;
  std::cout << ref << std::endl;
}
```

```
109
109
```

---

10. (10 points) Convert class Clock into a "Gang of Four" Singleton. The converted program should compile, link, and execute.

```
#include <iostream>

class Clock {
public:
  Clock() : ticks(0) {}
  int getTicks() const { return ticks; }
  void update() { ++ticks; }
private:
  int ticks;
};

int main( ) {
  Clock clock;
  clock.update();
  std::cout << clock.getTicks() << std::endl;
}
```

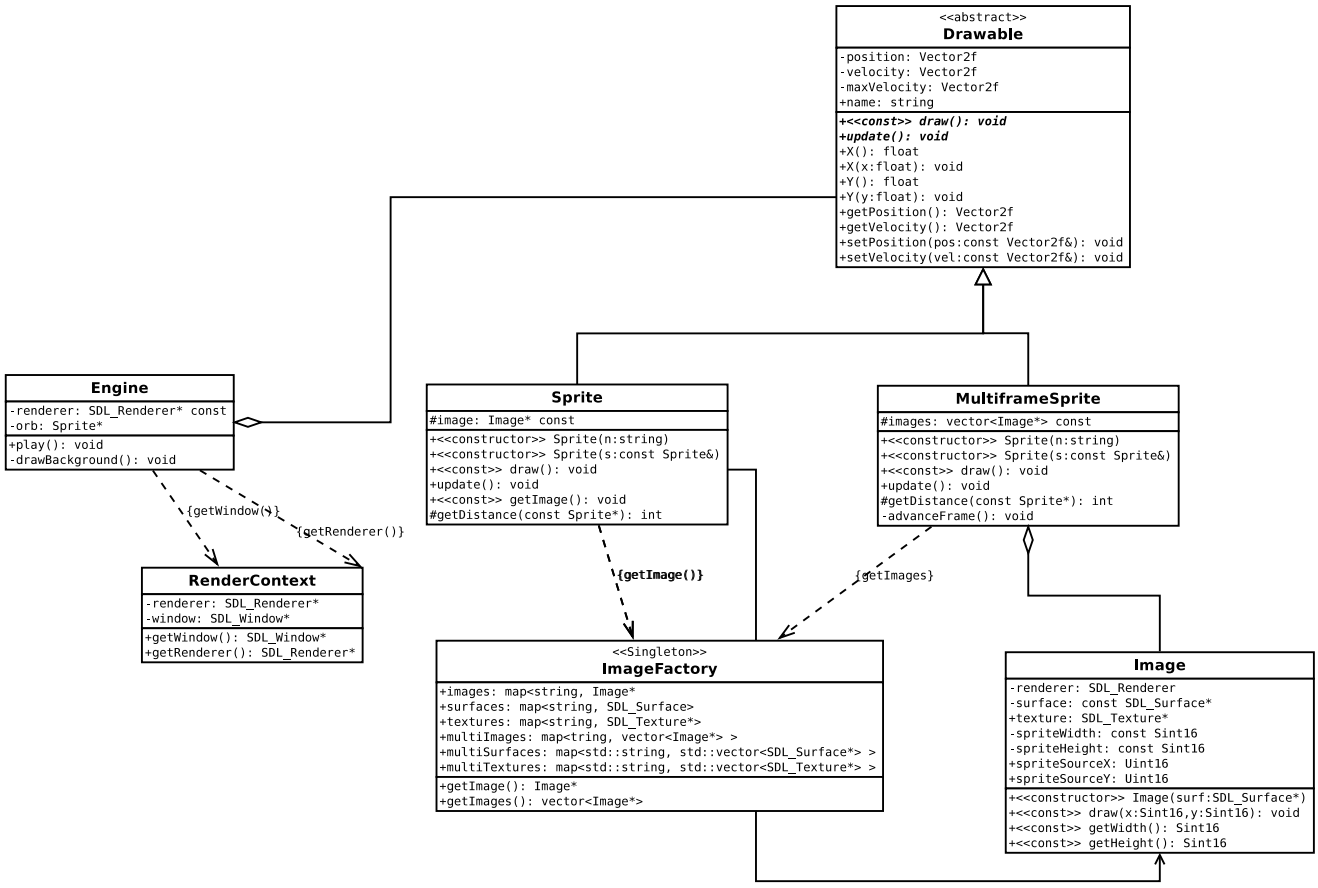

Figure 1: **Diagram of a class framework**.

---

11. (5 points) There is an edge in Figure 1 from class Engine to class Drawable:

    - what kind of edge is this and what does it indicate?
    - Write a declaration for Engine, using std::list that illustrates this edge.

    The line from Engine to Drawable illustrates aggregation, so that Engine contains an aggregation of Drawable -- because we want polymorphism in C++ it should be Drawable*

    The list declaration is:

    ```
    std::list<Drawable*> shapes;
    ```