1.

```
 8  ********************************************************************************/
 9  #include <iostream>
10  #include <string>
11  using namespace std;
12
13  // Base class
14  class Vehicle {
15  private:
16      string manufacturer;
17      int yearBuilt;
18
19  protected:
20      string model;
21
22  public:
23      Vehicle(string manu, int year, string mod) : manufacturer(manu), yearBuilt(year), model(mod) {}
24
25      void displayInfo() {
26          cout << "Manufacturer: " << manufacturer << endl;
27          cout << "Year Built: " << yearBuilt << endl;
28          cout << "Model: " << model << endl;
29      }
30  };
31
32  // Derived class
33  class Car : public Vehicle {
34  public:
35      int numDoors;
36
37      Car(string manu, int year, string mod, int doors) : Vehicle(manu, year, mod), numDoors(doors) {}
38
39      void displayCarInfo() {
40          displayInfo(); // Call the base class function to display manufacturer, year, and model
41          cout << "Number of Doors: " << numDoors << endl;
42      }
```

```
    }
};

int main() {
    // Create an instance of Car
    Car myCar("Toyota", 2021, "Camry", 4);

    // Display car information
    myCar.displayCarInfo();

    return 0;
}
```

input

```
Manufacturer: Toyota
Year Built: 2021
Model: Camry
Number of Doors: 4


...Program finished with exit code 0
Press ENTER to exit console.
```

2.

```cpp
#include <iostream>
#include <string>
using namespace std;

// Base class
class Shape {
protected:
    string color;

public:
    // Constructor to initialize color
    Shape(string col) : color(col) {}

    void displayShape() {
        cout << "Shape's color: " << color << endl;
    }
};

// Derived class Circle
class Circle : public Shape {
private:
    double radius;

public:
    // Constructor to initialize color and radius
    Circle(string col, double rad) : Shape(col), radius(rad) {}

    void displayCircle() {
        displayShape(); // Call the base class function to display color
        cout << "Circle's radius: " << radius << endl;
    }
};

// Derived class Rectangle
class Rectangle : public Shape {
```

```
44  private:
45      double width;
46      double height;
47
48  public:
49      // Constructor to initialize color, width, and height
50      Rectangle(string col, double w, double h) : Shape(col), width(w), height(h) {}
51
52      void displayRectangle() {
53          displayShape(); // Call the base class function to display color
54          cout << "Rectangle's width: " << width << endl;
55          cout << "Rectangle's height: " << height << endl;
56      }
57  };
58
59  int main() {
60      // Create an instance of Circle
61      Circle circle("Red", 5.5);
62      // Display circle information
63      circle.displayCircle();
64
65      // Create an instance of Rectangle
66      Rectangle rectangle("Blue", 4.0, 3.0);
67      // Display rectangle information
68      rectangle.displayRectangle();
69
70      return 0;
71  }
72
```

```
Shape's color: Red
Circle's radius: 5.5
Shape's color: Blue
Rectangle's width: 4
Rectangle's height: 3
```

3.

```cpp
#include <iostream>
using namespace std;

// Base class for logging
class BaseLogger {
public:
    BaseLogger() {
        cout << "BaseLogger constructed\n";
    }
    virtual ~BaseLogger() {
        cout << "BaseLogger destroyed\n";
    }
};

// Derived class for file logging
class FileLogger : public BaseLogger {
public:
    FileLogger() {
        cout << "FileLogger constructed\n";
    }
    ~FileLogger() {
        cout << "FileLogger destroyed\n";
    }
};

// Derived class for network logging
class NetworkLogger : public BaseLogger {
public:
    NetworkLogger() {
        cout << "NetworkLogger constructed\n";
    }
    ~NetworkLogger() {
        cout << "NetworkLogger destroyed\n";
```

```
    }
};

// Class that manages logging
class Application {
public:
    FileLogger fileLogger;
    NetworkLogger networkLogger;

    Application() {
        cout << "Application constructed\n";
    }
    ~Application() {
        cout << "Application destroyed\n";
    }
};

// Global instance of FileLogger
FileLogger globalFileLogger;

// Function that creates a static instance of NetworkLogger
void logNetworkEvent() {
    static NetworkLogger staticNetworkLogger;
    cout << "Logging network event\n";
}

int main() {
    cout << "Entering main\n";

    // Local instance of BaseLogger
    BaseLogger localBaseLogger;

    // Dynamic instance of Application
```

```
    // Dynamic instance of Application
    Application* app = new Application();

    // Call function to demonstrate static instance creation
    logNetworkEvent();

    // Ensure dynamic instance is properly deleted
    delete app;

    cout << "Exiting main\n";
    return 0;
}
```

```
BaseLogger constructed
FileLogger constructed
Entering main
BaseLogger constructed
BaseLogger constructed
FileLogger constructed
BaseLogger constructed
NetworkLogger constructed
Application constructed
BaseLogger constructed
NetworkLogger constructed
Logging network event
Application destroyed
NetworkLogger destroyed
BaseLogger destroyed
FileLogger destroyed
BaseLogger destroyed
Exiting main
BaseLogger destroyed
NetworkLogger destroyed
BaseLogger destroyed
FileLogger destroyed
BaseLogger destroyed
```