7.9 Choosing classes to create

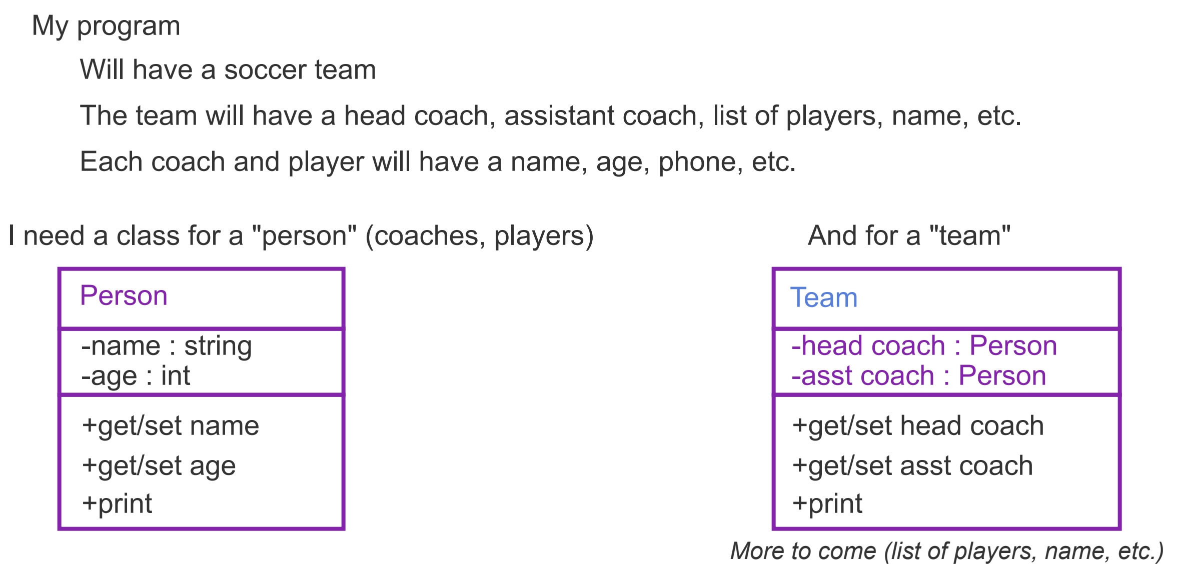
Decomposing into classes

Creating a program may start by a programmer deciding what "things" exist, and what each thing contains and does.

Below, the programmer wants to maintain a soccer team. The programmer realizes the team will have people, so decides to sketch a Person class. Each Person class will have private (shown by "-") data like name and age, and public (shown by "+") functions like get/set name, get/set age, and print. The programmer then sketches a Team class, which uses Person objects.

PARTICIPATION ACTIVITY 7.9.1: Creating a program by first sketching classes.

1 2 3 2x speed


My program

Will have a soccer team

The team will have a head coach, assistant coach, list of players, name, etc.

Each coach and player will have a name, age, phone, etc.

The programmer then sketches a Team class. Private items are head coach and asst coach, both of Person type. Public items are getters/setters and print.

Feedback?

PARTICIPATION ACTIVITY 7.9.2: Decomposing a program into classes.

Consider the example above.

1) Only one way exists to decompose a program into classes.

True [✓ False]

Correct

Thinking of a program as classes is sometimes helpful, sometimes not, depending on the application. Even when helpful, many ways exist to decompose the program into classes.

2) The - indicates a class' private item.

[✓ True] False

Correct

The - is a common convention for private items.

3) The + indicates additional private items.

True [✓ False]

Correct

The + is a common convention for public items, not private items.

4) The Team class uses the Person class.

[✓ True] False

Correct

Team has two private data items of type Person, namely the head coach, and asst coach.

5) The Person class uses the Team class.

True [✓ False]

Correct

Person has string and int data items, but no Team item.

Feedback?

Coding the classes

A programmer can convert the class sketches above into code. The programmer likely would first create and test the Person class, followed by the Team class.

Figure 7.9.1: SoccerTeam and TeamPerson classes.

TeamPerson.h	TeamPerson.cpp

```
#ifndef TEAMPERSON_H
#define TEAMPERSON_H

#include <string>
using namespace std;

class TeamPerson {
   public:
      void   SetFullName(string
firstAndLastName);
      void   SetAgeYears(int ageInYears);
      string GetFullName() const;
      int    GetAgeYears() const;
      void   Print() const;

   private:
      string fullName;
      int    ageYears;
};

#endif
```

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

#include "TeamPerson.h"

void TeamPerson::SetFullName(string
firstAndLastName) {
   fullName = firstAndLastName;
}

void TeamPerson::SetAgeYears(int
ageInYears) {
   ageYears = ageInYears;
}

string TeamPerson::GetFullName() const {
   return fullName;
}

int TeamPerson::GetAgeYears() const {
   return ageYears;
}

void TeamPerson::Print() const {
   cout << "Full name: "   << fullName
<< endl;
   cout << "Age (years): " << ageYears
<< endl;
}
```

SoccerTeam.h

```
#ifndef SOCCERTEAM_H
#define SOCCERTEAM_H

#include "TeamPerson.h"

class SoccerTeam {
   public:
      void SetHeadCoach(TeamPerson
teamPerson);
      void SetAssistantCoach (TeamPerson
teamPerson);

      TeamPerson GetHeadCoach() const;
      TeamPerson GetAssistantCoach() const;

      void Print() const;

   private:
      TeamPerson headCoach;
      TeamPerson assistantCoach;
      // Players omitted for brevity
};

#endif
```

SoccerTeam.cpp

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

#include "SoccerTeam.h"

void SoccerTeam::SetHeadCoach(TeamPerson
teamPerson) {
   headCoach = teamPerson;
}

void
SoccerTeam::SetAssistantCoach(TeamPerson
teamPerson) {
   assistantCoach = teamPerson;
}

TeamPerson SoccerTeam::GetHeadCoach()
const {
   return headCoach;
}

TeamPerson
SoccerTeam::GetAssistantCoach() const {
   return assistantCoach;
}

void SoccerTeam::Print() const {
   cout << "HEAD COACH: " << endl;
   headCoach.Print();
   cout << endl;

   cout << "ASSISTANT COACH: " << endl;
   assistantCoach.Print();
   cout << endl;
}
```

main.cpp

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

#include "SoccerTeam.h"
#include "TeamPerson.h"

int main() {
   SoccerTeam teamCalifornia;
   TeamPerson headCoach;
   TeamPerson asstCoach;

   headCoach.SetFullName("Mark Miwerds");
   headCoach.SetAgeYears(42);
   teamCalifornia.SetHeadCoach(headCoach);

   asstCoach.SetFullName("Stanley Lee");
   asstCoach.SetAgeYears(30);

teamCalifornia.SetAssistantCoach(asstCoach);

   teamCalifornia.Print();

   return 0;
}
```

```
HEAD COACH:
Full name: Mark Miwerds
Age (years): 42

ASSISTANT COACH:
Full name: Stanley Lee
Age (years): 30
```

Feedback?

PARTICIPATION ACTIVITY | 7.9.3: Coding classes. ✓

Consider the example above.

1) The programmer first sketched the desired classes, before writing the code seen above.

[✓ True] False

Correct

Planning a program before writing code is good practice.

2) The programmer wrote one large file containing all the classes.

True [✓ False]

Correct

Each class typically gets its own file.

3) Good practice would be to first write the TeamPerson class and then test that class, followed by writing the SoccerTeam class and testing that class.

[✓ True] False

Correct

Though not shown above, each class should be tested after being written.

Feedback?

Included files

Above, note that each file only includes needed header files. SoccerTeam.h has a TeamPerson member so includes TeamPerson.h. SoccerTeam.cpp includes SoccerTeam.h. main.cpp declares objects of both types so also includes both .h files. A common error is to include unnecessary .h files, which misleads the reader.

Note that only .h files are included, never .cpp files.

PARTICIPATION ACTIVITY | 7.9.4: Classes and includes. ✓

Consider the earlier SoccerTeam and TeamPerson classes. Indicate which .h files should be included in each file.

1) TeamPerson.h

- ○ TeamPerson.h
- ○ SoccerTeam.h
- [✓ No .h file needed]

Correct

TeamPerson.h doesn't have any types that need a .h file.

2) TeamPerson.cpp

- [✓ TeamPerson.h]
- ○ SoccerTeam.h
- ○ No .h file needed

Correct

TeamPerson.h has the class definition needed by TeamPerson.cpp.

3) SoccerTeam.h

- [✓ TeamPerson.h]
- ○ SoccerTeam.h
- ○ No .h file needed

Correct

The SoccerTeam class has data items of type TeamPerson, so needs TeamPerson's class definition.

4) SoccerTeam.cpp

- ○ TeamPerson.h
- [✓ SoccerTeam.h]
- ○ TeamPerson.cpp
- ○ TeamPerson.h and SoccerTeam.h

Correct

The file has SoccerTeam and TeamPerson types, and SoccerTeam.h includes TeamPerson.h.

5) main.cpp

- ○ main.h
- ○ TeamPerson.h
- [✓ TeamPerson.h and SoccerTeam.h]
- ○ TeamPerson.cpp
- ○ SoccerTeam.cpp

Correct

The file has TeamPerson and SoccerTeam types.

Feedback?