Programming Design In-class Practices

Templates, Vectors, and Exceptions

Ling-Chieh Kung

Department of Information Management
National Taiwan University

Problem 1: MyVector2D

- Consider the class **MyVector2D**:

```
class MyVector2D
{
private:
  int x;
  int y;
public:
  MyVector2D() : x(0), y(0) {}
  MyVector2D(int x, int y) : x(x), y(y) {}
  void print() const;
  bool operator==(const MyVector2D& v) const;
};
```

```
void MyVector2D::print() const
{
  cout << "(" << this->x
       << ", " << this->y << ")\n";
}
bool MyVector2D::operator==
  (const MyVector2D& v) const
{
  if(this->x == v.x
     && this->y == v.y)
    return true;
  else
    return false;
}
```

Problem 1: MyVector2D

- What will be the outcome of executing the following program?

```
int main()
{
  MyVector2D u(1.2, 2.3);
  MyVector2D v(1.4, 2.6);
  u.print();
  v.print();

  if(u == v)
    cout << "Equal!\n";
  else
    cout << "Unequal!\n";

  return 0;
}
```

Problem 2: MyVector2D with a template

- Modify **MyVector2D**:
  - Make the type of the two coordinates governed by a template.
  - What will be the outcome of executing the following two programs?

```
int main()
{
  MyVector2D<int> u(1.2, 2.3);
  MyVector2D<int> v(1.4, 2.6);
  u.print();
  v.print();

  if(u == v)
    cout << "Equal!\n";
  else
    cout << "Unequal!\n";

  return 0;
}
```

```
int main()
{
  MyVector2D<double> u(1.2, 2.3);
  MyVector2D<double> v(1.4, 2.6);
  u.print();
  v.print();

  if(u == v)
    cout << "Equal!\n";
  else
    cout << "Unequal!\n";

  return 0;
}
```

Problem 2: MyVector2D with a template

- Modify **MyVector2D**:
  - How to allow the following program to be executed correctly?

```
int main()
{
  MyVector2D<int> u(1.2, 2.3);
  MyVector2D<double> v(1.4,
2.6);
  u.print();
  v.print();

  if(u == v)
    cout << "Equal!\n";
  else
    cout << "Unequal!\n";

  return 0;
}
```

Problem 3: EntityArray

- Consider the class **Entity**, **Car**, **Passenger**, and **EntityArray**:

```
class Entity
{
protected:
  string id;
  bool isOn;
  bool isSer;
  double lon;
  double lat;
public:
  Entity(string id, bool isOn, bool isSer, double lon, double lat);
  void print();
};
```

Problem 3: EntityArray

- Consider the class **Entity**, **Car**, **Passenger**, and **EntityArray**:

```
Entity::Entity(string id, bool isOn, bool isSer, double lon, double lat)
  : id(id), isOn(isOn), isSer(isSer), lon(lon), lat(lat)
{
}

void Entity::print()
{
  cout << this->id << ": " << this->isOn << " " << this->isSer
       << " (" << this->lon << ", " << this->lat << ")" << endl;
}
```

Problem 3: EntityArray

- Consider the class **Entity**, **Car**, **Passenger**, and **EntityArray**:

```
class Car : public Entity
{
private:
public:
  Car(string id, bool isOn, bool isSer,
      double lon, double lat);
  void print();
};

Car::Car(string id, bool isOn, bool isSer,
         double lon, double lat)
  : Entity(id, isOn, isSer, lon, lat)
{
}
```

```
void Car::print()
{
  cout << this->id << ": ";
  if(this->isOn == true)
  {
    if(this->isSer == true)
      cout << "in-service, (";
    else
      cout << "empty, (";
    cout << this->lon << ", "
         << this->lat << ")";
  }
  else
    cout << "offline";
  cout << endl;
}
```

Problem 3: EntityArray

- Consider the class **Entity**, **Car**, **Passenger**, and **EntityArray**:

```
class Passenger : public Entity
{
private:
public:
  Passenger(string id, bool isOn, bool isSer,
            double lon, double lat);
  void print();
};

Passenger::Passenger(string id, bool isOn,
                     bool isSer,
                     double lon, double lat)
  : Entity(id, isOn, isSer, lon, lat)
{
}
```

```
void Passenger::print()
{
  cout << this->id << ": ";
  if(this->isOn == true)
  {
    if(this->isSer == true)
      cout << "in-service, (";
    else
      cout << "waiting, (";
    cout << this->lon << ", "
         << this->lat << ")";
  }
  else
    cout << "offline";
  cout << endl;
}
```

Problem 3: EntityArray

- Consider the class **Entity**, **Car**, **Passenger**, and **EntityArray**:

```
class EntityArray
{
protected:
  int capacity;
  int cnt;
  Entity** entityPtr;
public:
  EntityArray();
  ~EntityArray();
  bool add(string id, bool isOn, bool isSer, double lon, double lat);
  void print();
};
```

Problem 3: EntityArray

- Consider the class **Entity**, **Car**, **Passenger**, and **EntityArray**:

```
EntityArray::EntityArray()
{
  this->cnt = 0;
  this->capacity = 20000;
  this->entityPtr = new Entity*[this->capacity];
}

EntityArray::~EntityArray()
{
  for(int i = 0; i < this->cnt; i++)
    delete this->entityPtr[i];
  delete [] this->entityPtr;
}
```

Problem 3: EntityArray

- Consider the class **Entity**, **Car**, **Passenger**, and **EntityArray**:

```
bool EntityArray::add(string id, bool isOn, bool isSer, double lon, double lat)
{
  if(this->cnt < this->capacity)
  {
    this->entityPtr[this->cnt] = new Entity(id, isOn, isSer, lon, lat);
    this->cnt++;
    return true;
  }
  else
    return false;
}

void EntityArray::print()
{
  for(int i = 0; i < this->cnt; i++)
    this->entityPtr[i]->print();
}
```

Problem 3: EntityArray

- **EntityArray** is not good. **Why**?

```
int main()
{
  EntityArray ca;
  ca.add("5HE-313", true, true, 0, 0);  // 5HE-313: 1 1 (0, 0)
  ca.add("LPA-039", true, false, 1, 1); // LPA-039: 1 0 (1, 1)
  ca.print();

  EntityArray pa;
  pa.add("B90705023", true, true, 0, 0);  // B90705023: 1 1 (0, 0)
  pa.add("R94725008", true, false, 1, 1); // R94725008: 1 0 (1, 1)
  pa.print();

  return 0;
}
```

Problem 4: `EntityArray` (template)

- **`EntityArray`** is not good.
- Last time we use **inheritance** and **polymorphism** to solve the problem.
  - Let **`CarArray`** and **`PassengerArray`** inherit **`EntityArray`**.
  - **`CarArray::add()`** instantiates a **`Car`** object, and **`PassengerArray::add()`** instantiates a **`Passenger`** object.
- Now let's solve the problem by using a **template**.
  - There is no need to modify **`Entity`**, **`Car`**, and **`Passenger`**.
  - Let's modify **`EntityArray`**.

Problem 4: `EntityArray` (template)

- Please modify **`EntityArray`** to make the following happen:

```
int main()
{
  EntityArray<Car> ca;
  ca.add("5HE-313", true, true, 0, 0);  // 5HE-313: in-service, (0, 0)
  ca.add("LPA-039", true, false, 1, 1); // LPA-039: empty, (1, 1)
  ca.print();

  EntityArray<Passenger> pa;
  pa.add("B90705023", true, true, 0, 0);  // B90705023: in-service, (0, 0)
  pa.add("R94725008", true, false, 1, 1); // R94725008: waiting, (1, 1)
  pa.print();

  return 0;
}
```

Problem 5: add() with an exception

- Recall our **add()** function in Problem 3 (which is not good):

```
bool EntityArray::add(string id, bool isOn, bool isSer, double lon, double lat)
{
  if(this->cnt < this->capacity)
  {
    this->entityPtr[this->cnt] = new Entity(id, isOn, isSer, lon, lat);
    this->cnt++;
    return true;
  }
  else
    return false;
}
```

- In practice, it is possible that the capacity limit is reached.
  - How to **enforce** the caller of **add()** to check the result and respond to it?

Problem 5: `add()` with an exception

- Let's throw an **exception** **`overflow_error`** from **`add()`**.
  - Modify the header of **`add()`** to indicate (and limit ourselves to) this.
  - Modify the returned value from **`bool`** to **`void`**.
  - Modify the body to throw something when necessary.
  - Set **`capacity`** to 1 to test your program.
  - Show that the program will be terminated if an exception is not caught.
  - Catch an exception in a way you like.
- Declare **`print()`** with **`noexcept`**.
  - Explain what that means.

Problem 6: average grades

- Let a user enters no more than 100 decimal grades and then print out the average grades of the grades that are nonzero.
  - The input process ends with a −1.
- If all grades are 0, print out 0.
- Write a function that takes an array of grades and the number of grades as parameters.
  - It returns the average if there is at least one nonzero grade.
  - It throws **`logic_error`** if there is no nonzero grade.
- Let the main function catch the **`logic_error`** and respond to it.

Problem 7: Car and Passenger vectors

- Recall our class **EntityArray** in Problem 4.
- An alternative way of implementing the class **EntityArray** is to use **vector**.

```
template <typename entityType>
class EntityArray
{
protected:
  vector<entityType> entities;
public:
  EntityArray();
  bool add(string id, bool isOn, bool isSer, double lon, double lat);
  void print();
};
```

Problem 7: Car and Passenger vectors

- Please modify **EntityArray** so that the same main function still works:

```
int main()
{
  EntityArray<Car> ca;
  ca.add("5HE-313", true, true, 0, 0);  // 5HE-313: in-service, (0, 0)
  ca.add("LPA-039", true, false, 1, 1); // LPA-039: empty, (1, 1)
  ca.print();

  EntityArray<Passenger> pa;
  pa.add("B90705023", true, true, 0, 0);  // B90705023: in-service, (0, 0)
  pa.add("R94725008", true, false, 1, 1); // R94725008: waiting, (1, 1)
  pa.print();

  return 0;
}
```