Programming Design In-class Practices

Inheritance and Polymorphism

Ling-Chieh Kung

Department of Information Management
National Taiwan University

Ride sharing

- Let’s implement a **ride-sharing platform** (like Uber, Lyft, and Didi)!
- In a ride-sharing platform, there are **cars** and **passengers**.
  - Each car has exactly one driver. To make our life easier, let’s ignore drivers.
- n cars are registered in the platform.
  - At any moment, a car is either **offline** or **online**.
  - If it is online, it is either **empty** or **in-service**.
- m passengers are registered in the platform.
  - At any moment, a passenger is either **offline** or **online**.
  - If she/he is online, she/he is either **waiting** or **in-service**.
  - A waiting customer has made a request, has been paired with a car, and is waiting for her/his car. An in-service customer is moving toward her/his destination in a car.
- The **longitude** and **latitude** of an online car or passenger are recorded.

Ride sharing

- To make our life easier, let's assume that n and m are known and fixed.
  - $1 \leq n \leq 10000$ and $1 \leq m \leq 10000$.
- The main task of our program is to **assign empty cars to passengers**.
  - Do this when a passenger requests for a car.
  - A passenger does this when she/he logs in.
  - Once we assign an empty car to a passenger, their status should become in-service and waiting, respectively.
  - When they meet, the passenger's status should become in-service.
  - When they get to the passenger's destination, their status should become empty and offline, respectively.
- A car may get online/offline by turning on/off the app.

Ride sharing

- To complete our program, let's make a design:
  - Let's have a class **`Entity`** as the parent class of two classes **`Car`** and **`Passenger`**.
  - **`Entity`** should contain the common information that should be recorded in **`Car`** and **`Passenger`**.
- Things that we need to keep tracking:
  - Car: plate number (a C++ string), status (an integer), longitude (a **`double`**), latitude (a **`double`**), assigned passenger (a **`Passenger`** pointer).
  - Passenger: cellphone number (a C++ string), status (an integer), longitude (a **`double`**), latitude (a **`double`**), assigned car (a **`Car`** pointer).

The class **Entity**

- We start by creating the class **Entity**.

```
class Entity
{
private:
  string id;
  bool isOn;
  bool isSer; // means nothing if isOn == false
  double lon;
  double lat;
public:
  Entity(string id, bool isOn, bool isSer, double lon, double lat);
  void print();
};
```

The class `Entity`

```
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;
}
```

The class **Entity**

- Let’s test our class **Entity**:

```
int main()
{
  Entity a("5HE-313", true, true, 0, 0);
  a.print();
  return 0;
}
```

- Look fine!

Problem 1: the class **Car**

- Now let’s create the class **Car** by **inheriting** from **Entity**. Your friend wrote the following definition, which does not work. Tell him how to fix the bug(s).

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

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

Problem 1: the class **Car**

- Let’s test our revised **Car** (and **Entity**).

```
int main()
{
  Car a("5HE-313", true, true, 0, 0);
  a.print();
  return 0;
}
```

- Look fine, but the printed information is not good.

Problem 2: `Car::print()`

- Please override the **`print()`** function:

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

  - Print out “in-service”, “empty”, or “offline” depending on the status.
  - Print out the location only if it is online.
- Test it with the main function in the previous page.

Problem 3: the class CarArray (part 1)

- There will be a lot of cars in our program. Let’s use an array to store them.

```
int main()
{
  Car* cars[20000] = {0};
  int carCnt = 0;
  cars[0] = new Car("5HE-313", true, true, 0, 0);
  carCnt++;
  cars[1] = new Car("LPA-039", true, false, 1, 1);
  carCnt++;
  for(int i = 0; i < carCnt; i++)
    cars[i]->print();

  return 0;
}
```

- Look fine, but may we do better?

Problem 3: the class **CarArray** (part 1)

- Let’s create a class **CarArray**. Your friend proposed the definition below:

```
class CarArray
{
private:
  int capacity; // initialize it to 20000; may change later
  int cnt;
  Car* cars;
public:
  CarArray();
  // CarArray(const CarArray& ca);  // not needed
  // operator=(const CarArray& ca); // in this problem
  ~CarArray();
  void add();
  void print();
};
```

Problem 3: the class CarArray (part 1)

- His plan is to create a dynamic **Car** array. His constructor is:

```
CarArray::CarArray()
{
  this->cnt = 0;
  this->capacity = 20000;
  this->cars = new Car[this->capacity];
}
```

- Please point out the flaws, if any, of this design or conclude that there is no flaw.

Problem 4: the class CarArray (part 2)

- Given your suggestion, your friend modified his design to get the following:

```
class CarArray
{
private:
  int capacity; // initialize it to 20000; may change later
  int cnt;
  Car** carPtr;
public:
  CarArray();
  // CarArray(const CarArray& ca);  // not needed
  // operator=(const CarArray& ea); // in this problem
  ~CarArray();
  void add();
  void print();
};
```

Problem 4: the class CarArray (part 2)

- His constructor and destructor now become

```
CarArray::CarArray()
{
  this->cnt = 0;
  this->capacity = 20000;
  this->carPtr = new Car*[this->capacity];
}

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

Problem 4: the class **CarArray** (part 2)

- Help him finish the functions **add()** and **print()**. Test them with the following main function:

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

  return 0;
}
```

Problem 5: the class `Passenger`

- Please create the class **`Passenger`** by **inheriting** from **`Entity`**.
- Override **`print()`**:
  - Print out “in-service”, “waiting”, or “offline” depending on the status.
  - Print out the location only if it is online.
- Test your program with the following main function:

```
int main()
{
  Car c("5HE-313", true, false, 0, 0);
  c.print();

  Passenger p("B90705023", true, false, 0, 0);
  p.print();

  return 0;
}
```

Problem 6: the class `PassengerArray`

- Okay, so now it's time for **`PassengerArray`**…
- We may copy **`CarArray`** and do some modifications. Is there a better way?
- Why not do **inheritance** again!
  - Create a class **`EntityArray`** by modifying **`CarArray`**.
  - Let **`CarArray`** and **`PassengerArray`** be two child classes of **`EntityArray`**.

Problem 6: the class PassengerArray

- Seems that we do not need to do anything:

```
class CarArray : public EntityArray
{
};

class PassengerArray : public EntityArray
{
};
```

Problem 6: the class PassengerArray

- Test your **CarArray** and **PassengerArray** with the following main function:

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

  PassengerArray pa;
  pa.add("B90705023", true, true, 0, 0);
  pa.add("R94725008", true, false, 1, 1);
  pa.print(); // it works

  return 0;
}
```

Problem 6: the class `PassengerArray`

- The only problem remains is about the information printed.
  - We want to see status strings, not just 0 and 1.
- Do something to make this happen!

Problem 7: Preventing `Entity` objects

- In our system, an entity should either be a car or a passenger.
- There should be no **`Entity`** objects.
- Do something to make this happen!