For refactoring 2:

- A list of the files and the method(s) you are changing.
    - Changed Files:
        - bus.cc
        - bus.h
    - Changed Methods
        - Created DockingProcess(Route * cur_route, float distance)
        - Created SkipStop(float distance)
- The location of the changes. Be sure you explain briefly what you are doing at each location.
    - In bus.cc
        - A new method called void Bus::SkipStop(float distance) and a new method called
          double Bus::DockingProcess(Route * cur_route, float distance) is added at bus.cc.
        - The new added function were declared in bus.h.
        - In Update(), remove the code that handles the situations when the bus arrives at a stop under the if statement at line 74(outbound) and 97(inbound) then substitute in the new method DockingProcess.
        - In DockingProcess(), remove the code that handles the situations of skipping stop under the if statement at line 79 then substitute in the new method SkipStop.

Update() Before Extraction	Update() After Extraction

void Bus::Update() { // using common Update format
... ...
if (out_distance_ <= 0 && out_distance_ >= -speed_) {
// arrives at a outbound stop
stop_arrived_at_ = outgoing_route_->GetDestinationStop();
int unloaded = UnloadPassenger();
if (outgoing_route_->IsAtEnd()) {
out_distance_ = -1000;
distance_remaining_ = 0;
} else {
int loaded = stop_arrived_at_->LoadPassengers(this);
outgoing_route_->NextStop();
out_distance_ = outgoing_route_->GetNextStopDistance();
	void Bus::Update() { // using common Update format
	... ...
	if (out_distance_ <= 0 && out_distance_ >= -speed_) {
	// arrives at an outbound stop
	out_distance_ = DockingProcess(outgoing_route_, out_distance_);
	
	} else if (in_distance_ <= 0 && in_distance_ >= -speed_) {
	in_distance_ = DockingProcess(incoming_route_, in_distance_);
	
	} else if (out_distance_ > -1) {
	... ...
	}
	UpdateBusData();

```
      next_stop_ = outgoing_route_->GetDestinationStop();

      if (!unloaded && !loaded) {  // skip this stop when no
one waiting/off
        double offset = speed_ - distance_remaining_;  // bus
should travel
        distance_remaining_ = out_distance_ - offset;  // more
distance.
        if (distance_remaining_ < 0)  // bus shouldn't skip
more than 1 stop
          distance_remaining_ = 0;
        } else {
        distance_remaining_ = out_distance_;  // normal
case.
      }
    }
    distances_between_ = out_distance_;
  } else if (in_distance_ <= 0 && in_distance_ >= -speed_) {
    stop_arrived_at_ =
incoming_route_->GetDestinationStop();
    int unloaded = UnloadPassenger();
    if (incoming_route_->IsAtEnd()) {
      in_distance_ = -1000;
      distance_remaining_ = 0;
    } else {
      int loaded = stop_arrived_at_->LoadPassengers(this);
      incoming_route_->NextStop();
      in_distance_ =
incoming_route_->GetNextStopDistance();
      next_stop_ = incoming_route_->GetDestinationStop();

      if (!unloaded && !loaded) {  // skip this stop when no
one waiting/off
        double offset = speed_ - distance_remaining_;  // bus
should travel
        distance_remaining_ = in_distance_ - offset;  // more
distance.
        if (distance_remaining_ < 0)  // bus shouldn't skip
more than 1 stop
          distance_remaining_ = 0;            } else {
        distance_remaining_ = in_distance_;  // normal case.
      }
    }
    distances_between_ = distance;
    in_distance_ = DockingProcess(incoming_route_,
in_distance_);
  } else if (out_distance_ > -1) {
                                    ... ...
  }
  UpdateBusData();
  Move();
}
```

```
  Move();
}
```

A brief explanation of how this improves my code:

- This refactoring extracts two methods from a long redundant code. So it not only makes the code more understandable and readable but also allows me to reuse existing code and avoids large-scale copy and paste the code. Besides that, if other programmers or myself need to make changes or debug the docking process, this will not only reduce the workload of modifications but also avoid potential errors caused by copy and paste.