Zeid Kootbally
University of Maryland
College Park, MD

zeidk@umd.edu

RWA-3 on Lectures 8&9: OOP, Inheritance, and Polymorphism

ENPM809Y : Fall 2020
Due **Thursday, November 5, 2020**

Contents

Introduction

- **Scenario**: Two mobile robots are tasked to navigate through a maze and pick up objects.
- **Objectives**: For this assignment, you are not asked to write code that will drive robots in a maze. You are tasked to develop **only the structure** of a project using object-oriented programming, inheritance, and polymorphism, that will be used in the final project to drive robots in a maze.

Robots

- The scenario uses two types of mobile robotic arms that can drive and pick up objects in a maze.
- The first robot belongs to the C++ class **LandBasedWheeled** (see example of this type of robot in Figure 1a).
- The second robot belongs to the C++ class **LandBasedTracked** (see example of this type of robot in Figure 1b).
- Both classes derive from the base class **LandBasedRobot**.

(a) Husky

(b) LT2-F Tactical Robot

Figure 1: Robots used in the maze problem.

Robot Class Specifications

- Class **LandBasedRobot**
  - Attributes:
    - `std::string name_`: Name of the robot.
    - `double speed_`: Driving speed of the robot.
    - `double width_`: Width of the base of the robot.
    - `double length_`: Length of the base of the robot.
    - `double height_`: Height of the base of the robot.
    - `double capacity_`: Payload of the arm.
    - `int x_`: X coordinate of the robot in the maze.
    - `int y_`: Y coordinate of the robot in the maze.
  - Methods:
    - `void GoUp(x_,y_)`: Move the robot up in the maze.
    - `void GoDown(x_,y_)`: Move the robot down in the maze.
    - `void TurnLeft(x_,y_)`: Move the robot left in the maze.
    - `void TurnRight(x_,y_)`: Move the robot right in the maze.
    - `void PickUp(std::string)`: Arm picks up an object.
    - `void Release(std::string)`: Arm releases an object.
- Class **LandBasedWheeled**
  - Attributes:
    - `int wheel_number`: Number of wheels mounted on the robot.
    - `std::shared_ptr<std::string> wheel_type`: Type of wheels mounted on the robot.
- Class **LandBasedTracked**
  - Attributes:
    - `std::shared_ptr<std::string> track_type`: Type of track mounted on the robot.

The Program

- We do not worry about objects that the robots must pick up.

Objectives

- Using single `public` inheritance, create the following classes:
  - **LandBasedRobot** is an **abstract class**.
    - **LandBasedWheel** is a **concrete class** and is derived from **LandBasedRobot**.
    - **LandBasedTrack** is a **concrete class** and is derived from **LandBasedRobot**.
- Attributes of the base class must be declared `protected`.
- Attributes of the derived classes must be declared `protected`.
- All methods in based and derived classes must be declared `public`.
- For each class
  - Method prototypes go in the class definition (.h)
  - Method definitions go in the class implementation (.cpp)
- Wrap all your classes with the namespace `rwa3`
- In the method definitions, **do not write any "useful code"**. You only need to **output which function is being called** when your program runs. Remember that you are **not** tasked to drive robots in a maze at this point. You first need to build the structure of your program that will be used in the future to control these robots.
  - Example:

```
rwa3::LandBasedRobot::GoUp(int x, int y){
   std::cout << "LandBasedRobot::GoUp is called\n";
}
```

- Write the constructor(s) and destructor for each class.
  - Make sure to call the base class constructors from the derived class constructors.
  - You will need to at least implement the following constructor for the base class:
    - `LandBasedRobot(std::string name, int x, int y)` where `name` is used to initialize the `name_` attribute, `x` is used to initialize the `x_` attribute, and `y` is used to initialize the `y_` attribute.
- Write accessors and mutators directly in class definitions (.h)
  - **Note**: I did not provide any prototype for mutators and accessors for the C++ classes, so you will need to write them. You will need to at least implement the following accessors:

```
int get_x();//--get the x coordinate of a robot
int get_y();//--get the y coordinate of a robot
```

- Check if it is necessary to write your own deep copy constructors for each class. Remember that a default copy constructor, which does shallow copy, is provided by the compiler if you do not write your own copy constructor.
- Modify your classes to allow dynamic binding (`virtual` methods).
  - Remember best practices for the use of `virtual` and `override` keywords in derived classes.
  - **Note**: If you are using method prototypes (which you should be doing), `virtual` and `override` are placed in the method prototypes, not in the method definitions.

```
//--lanbasedtracked.h
class LandBasedTracked : public LandBasedRobot{
public:
   virtual void GoUp(int, int) override;
};

//--landbasedtracked.cpp
void rwa3::LandBasedTracked::GoUp(int x, int y)
{
   std::cout << "LandBasedTracked::GoUp is called\n";
}
```

- **Note**: You can use UML tools (e.g., https://circle.visual-paradigm.com/) to create your class diagrams and then generate C++ code. You will still probably need to edit the generated code.

The Main File

- You will need to test your program with the following code in **main.cpp**

Main file Code

```
#include "landbasedwheeled.h"
#include "landbasedtracked.h"
#include <vector>
#include <memory>
#include <iostream>
//--prototype
void FollowActionPath(std::shared_ptr<rwa3::LandBasedRobot> robot,
const std::vector<std::string> &vec, std::string obj);

void FollowActionPath(std::shared_ptr<rwa3::LandBasedRobot> robot,
const std::vector<std::string> &vec,std::string obj){
    int x{robot->get_x()};//--should be 1 for wheeled and 2 for tracked
    int y{robot->get_y()};//--should be 4 for wheeled and 3 for tracked
    for (auto s: vec){
        if (s.compare("up")==0)
            robot->GoUp(x,y);
        else if (s.compare("down")==0)
            robot->GoDown(x,y);
        else if (s.compare("right")==0)
            robot->TurnRight(x,y);
        else if (s.compare("left")==0)
            robot->TurnLeft(x,y);
        else if (s.compare("pickup")==0)
            robot->PickUp(obj);
        else if (s.compare("release")==0)
            robot->Release(obj);
    }
}

int main(){
    //--the following should throw an error since LandBasedRobot is an abstract class
    //rwa3::LandBasedRobot base_robot("none",1,2);

    //--pointer to base class for dynamic binding
    std::shared_ptr<rwa3::LandBasedRobot> wheeled =
    std::make_shared<rwa3::LandBasedWheeled>("Husky",1,4);

    std::vector<std::string> action_path_wheeled {"up","right","pickup","left","down",
    "release"};
    FollowActionPath(wheeled,action_path_wheeled,"book");
    std::cout << "--------------------------------------------------------------------\n";

    //--pointer to base class for dynamic binding
    std::shared_ptr<rwa3::LandBasedRobot> tracked =
    std::make_shared<rwa3::LandBasedTracked>("LT2-F",2,3);

    std::vector<std::string> action_path_tracked {"up","left","pickup","down","right",
    "release"};
    FollowActionPath(tracked,action_path_tracked,"cube");

    return 0;
}
```

- This is the expected output:

Main file Code

```
LandBasedWheeled::GoUp is called
LandBasedWheeled::TurnRight is called
LandBasedWheeled::PickUp is called
LandBasedWheeled::TurnLeft is called
LandBasedWheeled::GoDown is called
LandBasedWheeled::Release is called
------------------------------------------------------------------------
LandBasedTracked::GoUp is called
LandBasedTracked::TurnLeft is called
LandBasedTracked::PickUp is called
LandBasedTracked::GoDown is called
LandBasedTracked::TurnRight is called
LandBasedTracked::Release is called
```

Packaging The Project

Instructions to Package The Project

1. Create a project named **RWA3-Group#** (where # is your group number).
2. Create the following folders in your IDE:
   - **LandBasedRobot**
   - **LandBasedWheeled**
   - **LandBasedTracked**
3. Create a class in each directory. For example, the class **LandBasedRobot** must be created in the **LandBasedRobot** directory.
4. Document your classes, methods, and attributes using Doxygen documentation.
5. In your project, create a directory named **doc** and place your Doxyfile in it.
   - Example: For group 1, your Doxyfile will be placed in **RWA3-Group1/doc/Doxyfile**
6. Compress your project and rename it using the proper format.
7. Upload it on Canvas.