Faculty of Science, Engineering and Technology

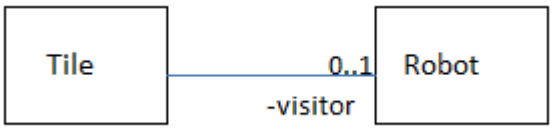
Object Oriented Programming

Robot Rally: C++ Implementation – Iteration 2

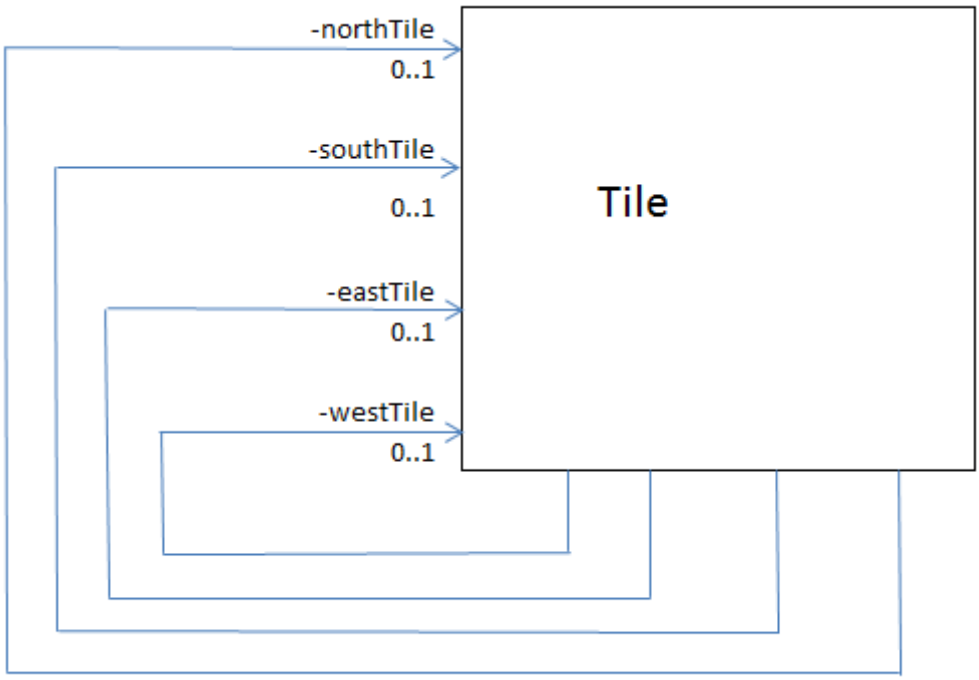
Outline

In this iteration we are going to focus on associations and collaboration between two objects.

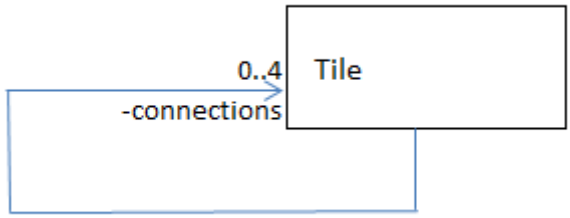
A Tile knows which Robot is visiting. This is an association and is depicted in the class diagram below. When a Robot visits a Tile the visitor attribute is set. It "points or refers" to the visiting Robot Object. The Tile may then work with the Robot Object (call its methods).



Tile(s) also collaborate with other Tile(s). This is depicted in the diagram below. Each Tile **may** have a connecting Tile in the north, south, east or west direction. This is an association between one Tile and another.



An alternative approach, that would use a collection, is depicted below. A collection of Tile could have an attribute that is implemented as a vector of Tile pointers.



The Tile class can be constructed using a default constructor, or by using a constructor that receives the visiting Robot.

The Tile class has a number of operations that relate to its attributes. These are shown in the diagram below. The Tile has the ability to say what Robot is currently visiting, the ability to set a new visiting Robot and the ability to remove a Robot.

The Tile class can add a new Tile at the North, South, East or West location. In the design, the attributes that know about the connected Tile(s) are named northTile, southTile, etc. A Tile can also return a reference to the Tile located at any of the four locations.

Tile
- visitor : Robot* - eastTile : Tile* - westTile : Tile* - northTile : Tile* - southTile : Tile*
+ Tile() + Tile(visitor : Robot*) + getTile (position : Direction) : Tile* + getVisitor() : Robot* + removeRobot(aVisitor : Robot*) : boolean + setTile (aTile : Tile*, position : Direction) + setVisitor(visitor : Robot*) + toString() : string

The following table outlines the Tile classification, and its responsibilities.

Classification	Responsibility	Details
Tile	Knows its Visiting Robot	A Tile may have one visiting Robot. At times a Tile will not have any visitors.
	Knows of other connected Tiles	A Tile may know of other Tile objects. The other Tiles may be in the North, South, East or West locations. Sometimes a location will not refer to any Tile.
	Can say which tile is connected at each of the Directions: North, South East and West.	When asked, a Tile will be able to return a reference to a Tile, based on a provided Direction. If no Tile is located at that Direction, or the Direction is invalid then nothing should be returned.
	Can receive details of a Tile and where it will be connected (The Direction) and update its knowledge of connected Tiles.	When the Tile is sent a reference to a Tile and a Direction, it will store that information in the appropriate attribute. If the direction is invalid or the Tile is not suitable then no change should be made to the attributes.
	Can assign a new Robot as a Visitor	The Tile can be sent a new Robot and set this Robot as the current visitor.
	Can remove a Robot as a Visitor	The Tile can remove the current Robot as a visitor. If the removal is successful the Tile will return true, otherwise the tile will return false.
	Can tell which Robot is visiting	When asked, the Tile can say which Robot is visiting
	Provide information about its state	When asked, the Tile can say whether it has a visitor or not, who the visitor is and say which locations are connected to other Tiles.
	Set visiting Robot at time of construction.	When building itself the Tile will have the ability to assign a Visiting Robot.

Refactoring the Robot Class

The Robot Class will now need some additional knowledge and behaviour.

A Robot must know which Tile it is on. The Robot must have the ability to set the Tile it is on and a user of a Robot may wish to know what Tile the Robot is currently on.

The Robot is now required to move. The move responsibility has been broken into two parts in this design. One method moves the robot one step and is private. The second method moves the Robot n steps (n should be -1,1,2 or 3) . The second method is public and will be called by other objects that wish the Robot to move. The diagram below shows the new design and the operations in **bold** will need to be coded.

(Use your Iteration 1 Robot as the starting point for this work)

Robot
- health : int - facing : Direction **- currentTile : Tile***
+ Robot() + Robot(face : Direction) - decrementHealth() : boolean + decrementHealth(int i) : int + getDirection() : Direction + getHealth() : int + rotate(turn : Rotate) : boolean + setDirection(face : Direction) **+ setCurrentTile(Tile : Tile*)** **+ getCurrentTile() : Tile*** **- moveOneSpot() : boolean** **+ move(steps: int)** + toString() : string

Classification	Responsibility	Details (Changes Only)
Robot	Knows which Tile it is on	The Robot will keep track of which Tile it is currently on.
	Has the ability to set which Tile it is on	A user of a Robot object will be able to set the Tile that a Robot is currently on
	Can report which tile it is on	A user of a Robot object will be able to ask the Robot which Tile it is on.
	Move one step (internal to this class, not for the outside world to use!)	The Robot will ask its currentTile for a connecting Tile that is located at the Direction the Robot is facing. If there is no connecting tile, the Robot will not move and false will be returned. If there is a connecting tile then the Robot will • remove itself from the current tile, • add itself to the new tile, • set its current tile to the new tile, and • return true if all works ok
	Move a number of steps	When asked, the Robot will try and move a number of steps. It will do this by calling the move one step method. The Robot must validate the number of steps before proceeding, If it is not a 1,2,3 or -1 then the Robot must not move. If any attempted move is unsuccessful then the Robot should stop trying to move. If the request is to move -1, then the Robot must perform a UTURN before attempting a move. In all circumstances where a move has failed then no further action should be taken.

Testing the Tile Class

You need to write code for a second class that will be used to Test the Tile Class

The code you write will be in a new Class called TestTile

The purpose of the tests in TestTile is to ensure that all public features of the Tile class work correctly.

The main method will call each of the tests.

Results of each test should be reported to the screen.

Note the use of the vector<Tile*> to represent an array of pointers to Tile instances; a variety of introductory tutorials on the use of the std::vector template can be found online; e.g. http://www.codeguru.com/

TestTile
- testRobot : Robot* - startTile : Tile*
+ main(args : vector<string>) - buildRobot() - buildTiles() : vector<Tile*> - connectTiles() : Tile* - placeRobotOnTile() - moveRobot(moves : int) : boolean - showResult(showThis : vector<Tile*>) - testSetUp() - testValidMove() -testInvalidStepCount() -testInvalidDirection() -testGetToEnd() -testTileGetMethods() -testTileSetMethods() -testWhenRobotNull()

Classification	Responsibility	Details
TestTile	Knows which Robot is being tested	The Robot will have one attribute that knows the Robot to be tested.
	Knows the Tile on which the Robot will first be placed.	In order to run this test a set of interlinked tiles will need to be created. However it is only necessary to know of the first tile in the set to run the test. The first Tile will be the one to which the Robot is assigned.
	Is executable	TestTile will have a main method from which all tests will be called. This is the method from which the program will start running.
	Can build a Robot	TestTile will be able to build a Robot, set it facing in a suitable direction and then assign it to an attribute.
	Can build the tiles needed to do a test.	Five Tiles must be built. The job here is to build the five tiles and to return a set of tiles.
	Can connect the Tiles together	The five tiles must be built into the shape of an “L”. The start Tile will be the most Northerly Tile. Connected at its South end will be a second Tile, connected at the second Tile’s south end will be a third Tile. Connected at the third Tiles East end will be a forth Tile. Connected at the forth Tiles East end will be a fifth Tile. The job here is to connect the tiles together and assign the start tile. (Note a n array of file will be passed to this method)
	Place Robot on Start Tile	This task requires that the start tile has been assigned and the Robot has been built. You must check this before you proceed further. The second part requires you to assign the Robot to the Start Tile.
	Move the Robot a number of steps	The task here is to move the robot a number of steps. The Robot should have been programmed to only move in the direction it is facing and to move either 1 step (-1) backwards or 1, 2 or 3 steps forward. This checking is done by the Robot. If the requested move is successful then true should be returned, if it is unsuccessful then false should be returned.
	TestTile must be able to show the current state of the Tiles and the Robot	When this behaviour is invoked, details of each Tile are displayed on the screen. This should show us where the Robot is, what direction it is facing, and how the tiles are connected. Enquiries need to be made of each Tile and the Robot (using the toString methods).

Classification	Responsibility	Details
	Test that the Tiles and Robot can be setup	Think about how each of the following will be achieved. Add the main steps here.
	Test a valid move	
	Test invalid move (step count wrong)	
	Test invalid move as (Robot facing in wrong direction)	
	Test Robot can be moved to last tile	
	Test getter methods in Tile	
	Test setter methods in Tile	
	Test for handling of null exception (Robot not set.)	

Modeling Interactions / Understanding how it will work.

Sequence diagrams are often used to understand how parts of a program will work. They include the objects involved and the messages sent between the objects that are required to achieve the task.

The syntax is described by the Unified Modeling Language. This is a dynamic diagram as it shows change over time. The sequence starts at the top left hand corner and proceeds down the page. Each solid line indicates a message being sent from one object to another. Each dashed line indicates that a method has finished running and control of the program is being returned to the object that first sent the message.

There are many tools for drawing sequence diagrams but often the best tool is a white board.

Things to consider before drawing a sequence diagram are

- Where will it start
- Where will it end
- What objects already exist
- Am I modeling for success or failure

In this case we want to model **move** Robot

- The move will probably start when the TestTile class calls the move method of Robot.
- The move will probably end when the Robot has been moved from one Tile to the next and control of the program has returned to the TestClass
- The objects that exist already will be the Robot and the set of Tile objects.
- I will assume that the Robot has already been placed on the startTile.
- I will assume that the Robot is facing South and it will be able to move
- I am going to model for a successful move

Think about the steps necessary to complete the move.

My initial thoughts are the Robot is asked to move

- The robot finds its current tile
- The robot gets the connecting tile on the south side
- The robot moves to the new tile
- The robot removes itself from its last tile.

Try and do the diagram now.
Start by drawing the required objects along the top of an A4 page.

Your tutor will work through this diagram in class in week 3.

You should include a hand drawn sequence diagram in your submission for Iteration 2.
Take a photo and add it to your submission.
Add a few comments on the side explaining what each of the symbols mean.

This link may be useful
http://www.ibm.com/developerworks/rational/library/3101.html