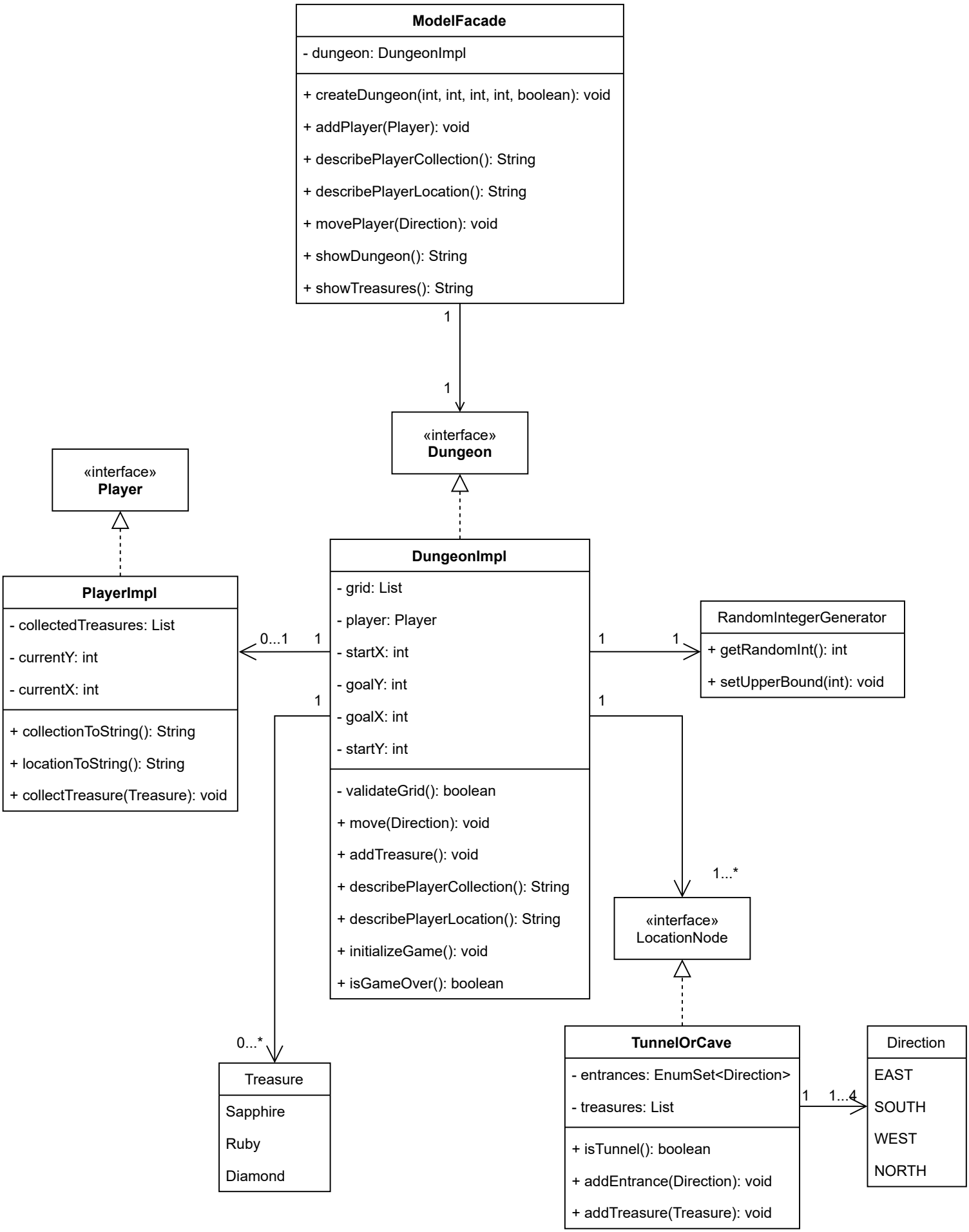
ModelFacade
- dungeon: DungeonImpl
+ createDungeon(int, int, int, int, boolean): void
+ addPlayer(Player): void
+ describePlayerCollection(): String
+ describePlayerLocation(): String
+ movePlayer(Direction): void
+ showDungeon(): String
+ showTreasures(): String
1
1
«interface»
Dungeon
«interface»
Player
DungeonImpl
- grid: List
- player: Player
- startX: int
- goalY: int
- goalX: int
- startY: int
- validateGrid(): boolean
+ move(Direction): void
+ addTreasure(): void
+ describePlayerCollection(): String
+ describePlayerLocation(): String
+ initializeGame(): void
+ isGameOver(): boolean
PlayerImpl
- collectedTreasures: List
- currentY: int
- currentX: int
+ collectionToString(): String
+ locationToString(): String
+ collectTreasure(Treasure): void
0...1
1
1
1
RandomIntegerGenerator
+ getRandomInt(): int
+ setUpperBound(int): void
1
1
1...*
«interface»
LocationNode
0...*
Treasure
Sapphire
Ruby
Diamond
TunnelOrCave
- entrances: EnumSet<Direction>
- treasures: List
+ isTunnel(): boolean
+ addEntrance(Direction): void
+ addTreasure(Treasure): void
1
1...4
Direction
EAST
SOUTH
WEST
NORTH

Testing design for RandomIntegerGenerator

Testing construction and fixed random seed	Input	Expected Value
Use fixed random seed	RandomIntegerGenerator(1, 5, 10)	4

Testing uniform distribution	Input	Expected Value
Experiment 1 million times and count the generated integer	RandomIntegerGenerator(0, 9)	The frequency of number from 0 ~ 9 should be equals to 10%

Testing lower bound larger than upper bound	Input	Expected Value
Construcor does not accept lower bound that larger than upper bound	RandomIntegerGenerator(5, 2)	IllegalArgumentException

Testing design for Treasure

```
// helper method
protected void getTreasure(String type) {
  ... // return corresponding treasure type
}
```

Testing double dispatch	Input	Expected Value
Check the type of Ruby	getTreasure("Ruby").isRuby()	True
Check the type of Ruby	getTreasure("Ruby").isDiamond()	False
Check the type of Ruby	getTreasure("Ruby").isRuby()	False
Check the type of Diamond	getTreasure("Diamond").isDiamond()	True
Check the type of Diamond	getTreasure("Diamond").isRuby()	False
Check the type of Diamond	getTreasure("Diamond").isSapphire()	False
Check the type of Sapphire	getTreasure("Sapphire").isSapphire()	True
Check the type of Sapphire	getTreasure("Sapphire").isRuby()	False
Check the type of Sapphire	getTreasure("Sapphire").isDiamond()	False

Testing design for TunnelOrCave

```
// helper method
protected void getTunnelOrCave(int nEntrance) {
  ... // return corresponding location with specific number of entrance
}
```

Testing addEntrance	Input	Expected Value
Add 1~4 entrance to the location	getTunnelOrCave(1~4)	The size of the entrances of the location should be 1~4
Add entrance to the location that has already contained 4 entrances	getTunnelOrCave(4).addEntrance(Direction.East)	IllegalStateException

Testing isTunnel	Input	Expected Value
Add 2 entrance to the location	getTunnelOrCave(2).isTunnel()	True
Add 1 entrance to the location	getTunnelOrCave(1).isTunnel()	False
Add 3 entrance to the location	getTunnelOrCave(3).isTunnel()	False
Add 4 entrance to the location	getTunnelOrCave(4).isTunnel()	False

Testing design for Player

Testing constructor	Input	Expected Value
Simply new a player and ust toString() to verify fields		"Collected treasures: [], Current Locatino: [0, 0]"

Testing collectTreasure()	Input	Expected Value
Collected treasure		"Collected treasures: [Diamond, Sapphire]"

Testing design for Dungeon

```
// helper method
protected void createDungeon(int length, int width, int interconnectivity, boolean
wrap, int percentageTreasureCaves) {
  ... // return a dungeon with input length and width and interconnectivity
}
```

Testing constructor	Input	Expected Value
Test Dungeon constructor	createDungeon(3, 4, 0, false, 0)	The grid of dungeon should has 4 caves in a row and 3 caves in a column
Test instantiate Dungeon with illegal distance between every two caves	createDungeon(3, 3, 0, false, 0)	IllegalArgumentException
Test illegal interconnectivity in wrap dungeon	createDungeon(3, 4, 25, true, 0)	IllegalArgumentException
Test illegal interconnectivity in dungeon	createDungeon(3, 4, 20, true, 0)	IllegalArgumentException
Test negative input length	createDungeon(-3, 4, 20, true, 0)	IllegalArgumentException
Test negative input width	createDungeon(3, -4, 20, true, 0)	IllegalArgumentException
Test negative input interconnectivity	createDungeon(3, 4, -20, true, 0)	IllegalArgumentException
Test negative input percentageTreasureCaves	createDungeon(3, 4, 20, true, -1)	IllegalArgumentException

Test addTreasure()	Input	Expected Value
Add treasures in at least 20% caves	createDungeon(-3, 4, 20, true, 20)	20% of caves have at least one treasure

Because description of players' collected treasures and location have been tested in the testing of `Player`, and `Dungeon` simply calls the method of `Player`, so there is no need to test description methods here.

Test initializeGame()	Input	Expected Value
Select start point and goal point for player	createDungeon(-3, 4, 20, true, 20).initializeGame()	The manhattan distance between start point and goal point should be larger than 5

Test isGameOver()	Input	Expected Value
Mocking the player has reached the goal point		True
Mocking the player is still on the way toward the goal point		False