Package dev.robocode.tankroyale.botapi

Class Bot

java.lang.Object
  dev.robocode.tankroyale.botapi.BaseBot
    dev.robocode.tankroyale.botapi.Bot

All Implemented Interfaces:

`IBaseBot`, `IBot`

```
public abstract class Bot
extends BaseBot
implements IBot
```

Abstract bot class provides convenient methods for movement, turning, and firing the gun. Most bots should inherit from this class.

Field Summary

Fields inherited from interface dev.robocode.tankroyale.botapi.IBaseBot

`MAX_NUMBER_OF_TEAM_MESSAGES_PER_TURN`, `TEAM_MESSAGE_MAX_SIZE`

Constructor Summary

Constructors

Modifier	Constructor	Description
`protected`	`Bot()`	
`protected`	`Bot(BotInfo botInfo)`	
`protected`	`Bot(BotInfo botInfo, java.net.URI serverUrl)`	
`protected`	`Bot(BotInfo botInfo, java.net.URI serverUrl, java.lang.String serverSecret)`	

All Methods Instance Methods Concrete Methods

Modifier and Type	Method	Description
void	**back**(double distance)	Moves the bot backward until it has traveled a specific distance from its current position, or it is moving into an obstacle.
void	**fire**(double firepower)	Fire the gun in the direction as the gun is pointing.
void	**forward**(double distance)	Moves the bot forward until it has traveled a specific distance from its current position, or it is moving into an obstacle.
double	**getDistanceRemaining**()	Returns the distance remaining till the bot has finished moving after having called `IBot.setForward(double)`, `IBot.setBack(double)`, `IBot.forward(double)`, or `IBot.back(double)`.
double	**getGunTurnRemaining**()	Returns the remaining turn in degrees till the gun has finished turning after having called `IBot.setTurnGunLeft(double)`, `IBot.setTurnGunRight(double)`, `IBot.turnGunLeft(double)`, or `IBot.turnGunRight(double)`.
double	**getRadarTurnRemaining**()	Returns the remaining turn in degrees till the radar has finished turning after having called `IBot.setTurnRadarLeft(double)`, `IBot.setTurnRadarRight(double)`, `IBot.turnRadarLeft(double)`, or `IBot.turnRadarRight(double)`.
double	**getTurnRemaining**()	Returns the remaining turn in degrees till the bot has finished turning after having called `IBot.setTurnLeft(double)`, `IBot.setTurnRight(double)`, `IBot.turnLeft(double)`, or `IBot.turnRight(double)`.
boolean	**isRunning**()	Checks if this bot is running.
void	**rescan**()	Scan (again) with the radar.

	(double distance)	a specific distance from its current position, or it is moving into an obstacle.
void	**setForward** (double distance)	Set the bot to move forward until it has traveled a specific distance from its current position, or it is moving into an obstacle.
void	**setGunTurnRate** (double turnRate)	Sets the turn rate of the gun, which can be positive and negative.
void	**setRadarTurnRate** (double turnRate)	Sets the turn rate of the radar, which can be positive and negative.
void	**setTargetSpeed** (double targetSpeed)	Sets the new target speed for the bot in units per turn.
void	**setTurnGunLeft** (double degrees)	Set the gun to turn to the left (following the increasing degrees of the unit circle) until it turned the specified amount of degrees.
void	**setTurnGunRight** (double degrees)	Set the gun to turn to the right (following the decreasing degrees of the unit circle) until it turned the specified amount of degrees.
void	**setTurnLeft** (double degrees)	Set the bot to turn to the left (following the increasing degrees of the unit circle) until it turned the specified amount of degrees.
void	**setTurnRadarLeft** (double degrees)	Set the radar to turn to the left (following the increasing degrees of the unit circle) until it turned the specified amount of degrees.
void	**setTurnRadarRight** (double degrees)	Set the radar to turn to the right (following the decreasing degrees of the unit circle) until it turned the specified amount of degrees.
void	**setTurnRate** (double turnRate)	Sets the turn rate of the bot, which can be positive and negative.
void	**setTurnRight** (double degrees)	Set the bot to turn to the right (following the decreasing degrees of the unit circle) until it turned the specified amount of degrees.
void	**stop**()	Stop all movement including turning the gun and radar.

	(double degrees)	degrees of the unit circle) until it turned the specified amount of degrees.
void	**turnGunRight** (double degrees)	Turn the gun to the right (following the decreasing degrees of the unit circle) until it turned the specified amount of degrees.
void	**turnLeft** (double degrees)	Turn the bot to the left (following the increasing degrees of the unit circle) until it turned the specified amount of degrees.
void	**turnRadarLeft** (double degrees)	Turn the radar to the left (following the increasing degrees of the unit circle) until it turned the specified amount of degrees.
void	**turnRadarRight** (double degrees)	Turn the radar to the right (following the increasing degrees of the unit circle) until it turned the specified amount of degrees.
void	**turnRight** (double degrees)	Turn the bot to the right (following the increasing degrees of the unit circle) until it turned the specified amount of degrees.
void	**waitFor** (**Condition** condition)	Blocks until a condition is met, i.e.

Methods inherited from class dev.robocode.tankroyale.botapi.BaseBot

addCustomEvent, broadcastTeamMessage, calcBulletSpeed, calcGunHeat, calcMaxTurnRate, clearEvents, getArenaHeight, getArenaWidth, getBodyColor, getBulletColor, getBulletStates, getDirection, getEnemyCount, getEnergy, getEventPriority, getEvents, getFirepower, getGameType, getGraphics, getGunColor, getGunCoolingRate, getGunDirection, getGunHeat, getGunTurnRate, getMaxGunTurnRate, getMaxInactivityTurns, getMaxRadarTurnRate, getMaxSpeed, getMaxTurnRate, getMyId, getNumberOfRounds, getRadarColor, getRadarDirection, getRadarTurnRate, getRoundNumber, getScanColor, getSpeed, getTargetSpeed, getTeammateIds, getTimeLeft, getTracksColor, getTurnNumber, getTurnRate, getTurnTimeout, getTurretColor, getVariant, getVersion, getX, getY, go, isAdjustGunForBodyTurn, isAdjustRadarForBodyTurn, isAdjustRadarForGunTurn, isDebuggingEnabled, isDisabled, isStopped, isTeammate, removeCustomEvent, sendTeamMessage, setAdjustGunForBodyTurn, setAdjustRadarForBodyTurn, setAdjustRadarForGunTurn, setBodyColor, setBulletColor, setEventPriority, setFire, setFireAssist, setGunColor, setInterruptible, setMaxGunTurnRate, setMaxRadarTurnRate, setMaxSpeed, setMaxTurnRate, setRadarColor, setRescan, clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods inherited from interface dev.robocode.tankroyale.botapi.IBaseBot

addCustomEvent, bearingTo, broadcastTeamMessage, calcBearing, calcBulletSpeed, calcDeltaAngle, calcGunBearing, calcGunHeat, calcMaxTurnRate, calcRadarBearing, clearEvents, directionTo, distanceTo, getArenaHeight, getArenaWidth, getBodyColor, getBulletColor, getBulletStates, getDirection, getEnemyCount, getEnergy, getEventPriority, getEvents, getFirepower, getGameType, getGraphics, getGunColor, getGunCoolingRate, getGunDirection, getGunHeat, getGunTurnRate, getMaxGunTurnRate, getMaxInactivityTurns, getMaxRadarTurnRate, getMaxSpeed, getMaxTurnRate, getMyId, getNumberOfRounds, getRadarColor, getRadarDirection, getRadarTurnRate, getRoundNumber, getScanColor, getSpeed, getTargetSpeed, getTeammateIds, getTimeLeft, getTracksColor, getTurnNumber, getTurnRate, getTurnTimeout, getTurretColor, getVariant, getVersion, getX, getY, go, gunBearingTo, isAdjustGunForBodyTurn, isAdjustRadarForBodyTurn, isAdjustRadarForGunTurn, isDebuggingEnabled, isDisabled, isStopped, isTeammate, normalizeAbsoluteAngle, normalizeRelativeAngle, onBotDeath, onBulletFired, onBulletHit, onBulletHitBullet, onBulletHitWall, onConnected, onConnectionError, onCustomEvent, onDeath, onDisconnected, onGameEnded, onGameStarted, onHitBot, onHitByBullet, onHitWall, onRoundEnded, onRoundStarted, onScannedBot, onSkippedTurn, onTeamMessage, onTick, onWonRound, radarBearingTo, removeCustomEvent, sendTeamMessage, setAdjustGunForBodyTurn, setAdjustRadarForBodyTurn, setAdjustRadarForGunTurn, setBodyColor, setBulletColor, setEventPriority, setFire, setFireAssist, setGunColor, setInterruptible, setMaxGunTurnRate, setMaxRadarTurnRate, setMaxSpeed, setMaxTurnRate, setRadarColor, setRescan, setResume, setScanColor, setStop, setStop, setTracksColor, setTurretColor, start

Methods inherited from interface dev.robocode.tankroyale.botapi.IBot

run

Constructor Detail

See Also:

`BaseBot()`

Bot

```
protected Bot(BotInfo botInfo)
```

See Also:

`BaseBot(BotInfo)`

Bot

```
protected Bot(BotInfo botInfo,
              java.net.URI serverUrl)
```

See Also:

`BaseBot(BotInfo, URI)`

Bot

```
protected Bot(BotInfo botInfo,
              java.net.URI serverUrl,
              java.lang.String serverSecret)
```

See Also:

`BaseBot(BotInfo, URI, String)`

Method Detail

setTurnRate

```
public final void setTurnRate(double turnRate)
```

the turn rate of the bot from the turn rate of the gun and radar. But be aware that the turn limits defined for the gun and radar cannot be exceeded.

The turn rate is truncated to `Constants.MAX_TURN_RATE` if the turn rate exceeds this value.

If this property is set multiple times, the last value set before `IBaseBot.go()` counts.

Specified by:

`setTurnRate` in interface `IBaseBot`

Overrides:

`setTurnRate` in class `BaseBot`

Parameters:

`turnRate` - is the new turn rate of the bot in degrees per turn.

setGunTurnRate

```
public final void setGunTurnRate(double turnRate)
```

Sets the turn rate of the gun, which can be positive and negative. The gun turn rate is measured in degrees per turn. The turn rate is added to the current turn direction of the gun. But it is also added to the current direction of the radar. This is because the radar is mounted on the gun, and hence moves with the gun. You can compensate for the turn rate of the gun by subtracting the turn rate of the gun from the turn rate of the radar. But be aware that the turn limits defined for the radar cannot be exceeded.

The gun turn rate is truncated to `Constants.MAX_GUN_TURN_RATE` if the gun turn rate exceeds this value.

If this property is set multiple times, the last value set before `IBaseBot.go()` counts.

Specified by:

`setGunTurnRate` in interface `IBaseBot`

Overrides:

`setGunTurnRate` in class `BaseBot`

Parameters:

`turnRate` - is the new turn rate of the gun in degrees per turn.

setRadarTurnRate

added to the radar direction, as the radar moves with the gun, which is mounted on the gun that moves with the body. You can compensate for the turn rate of the gun by subtracting the turn rate of the bot and gun from the turn rate of the radar. But be aware that the turn limits defined for the radar cannot be exceeded.

The radar turn rate is truncated to `Constants.MAX_RADAR_TURN_RATE` if the radar turn rate exceeds this value.

If this property is set multiple times, the last value set before `IBaseBot.go()` counts.

Specified by:

`setRadarTurnRate` in interface `IBaseBot`

Overrides:

`setRadarTurnRate` in class `BaseBot`

Parameters:

`turnRate` - is the new turn rate of the radar in degrees per turn.

isRunning

```
public final boolean isRunning()
```

Checks if this bot is running.

Specified by:

`isRunning` in interface `IBot`

Returns:

`true` when the bot is running, `false` otherwise.

setTargetSpeed

```
public final void setTargetSpeed(double targetSpeed)
```

Sets the new target speed for the bot in units per turn. The target speed is the speed you want to achieve eventually, which could take one to several turns depending on the current speed. For example, if the bot is moving forward with max speed, and then must change to move backward at full speed, the bot will have to first decelerate/brake its positive speed (moving forward). When passing speed of zero, it will then have to accelerate back to achieve max negative speed.

value.

If this property is set multiple times, the last value set before `IBaseBot.go()` counts.

Specified by:

`setTargetSpeed` in interface `IBaseBot`

Overrides:

`setTargetSpeed` in class `BaseBot`

Parameters:

`targetSpeed` - is the new target speed in units per turn.

setForward

`public final void setForward(double distance)`

Set the bot to move forward until it has traveled a specific distance from its current position, or it is moving into an obstacle. The speed is limited by `IBaseBot.setMaxSpeed(double)`.

When the bot is moving forward, the `Constants.ACCELERATION` determines the acceleration of the bot that adds 1 additional unit to the speed per turn while accelerating. However, the bot is faster at braking. The `Constants.DECELERATION` determines the deceleration of the bot that subtracts 2 units from the speed per turn.

This method will first be executed when `IBaseBot.go()` is called making it possible to call other set methods before execution. This makes it possible to set the bot to move, turn the body, radar, gun, and also fire the gun in parallel in a single turn when calling `IBaseBot.go()`. But notice that this is only possible to execute multiple methods in parallel by using **setter** methods only prior to calling `IBaseBot.go()`.

If this method is called multiple times, the last call before `IBaseBot.go()` is executed, counts.

This method will cancel the effect of prior calls to `IBaseBot.setTargetSpeed(double)` as the setForward and `IBot.setBack(double)` methods calls the `IBaseBot.setTargetSpeed(double)` for each turn until `IBot.getDistanceRemaining()` reaches 0.

Specified by:

`setForward` in interface `IBot`

Parameters:

`distance` - is the distance to move forward. If negative, the bot will move backward. If `Double.POSITIVE_INFINITY` the bot will move forward infinitely. If

forward

`public final void forward(double distance)`

Moves the bot forward until it has traveled a specific distance from its current position, or it is moving into an obstacle. The speed is limited by `IBaseBot.setMaxSpeed(double)`.

When the bot is moving forward, the `Constants.ACCELERATION` determine the acceleration of the bot that adds 1 additional unit to the speed per turn while accelerating. However, the bot is faster at braking. The `Constants.DECELERATION` determines the deceleration of the bot that subtracts 2 units from the speed per turn.

This call is executed immediately by calling `IBaseBot.go()` in the code behind. This method will block until it has been completed, which can take one to several turns. New commands will first take place after this method is completed. If you need to execute multiple commands in parallel, use **setter** methods instead of this blocking method.

This method will cancel the effect of prior calls to `IBaseBot.setTargetSpeed(double)`, `IBot.setForward(double)`, and `IBot.setBack(double)` methods.

Specified by:

`forward` in interface `IBot`

Parameters:

`distance` - is the distance to move forward. If negative, the bot will move backward. If `Double.POSITIVE_INFINITY` the bot will move forward infinitely. If `Double.NEGATIVE_INFINITY` the bot will move backward infinitely.

See Also:

`IBot.setForward(double)`, `IBot.setBack(double)`, `IBot.back(double)`, `IBot.getDistanceRemaining()`, `IBaseBot.setTargetSpeed(double)`

setBack

`public final void setBack(double distance)`

Set the bot to move backward until it has traveled a specific distance from its current position, or it is moving into an obstacle. The speed is limited by `IBaseBot.setMaxSpeed(double)`.

When the bot is moving forward, the `Constants.ACCELERATION` determines the acceleration of the bot that adds 1 additional unit to the speed per turn while accelerating.

body, radar, gun, and also fire the gun in parallel in a single turn when calling `IBaseBot.go()`. But notice that this is only possible to execute multiple methods in parallel by using **setter** methods only prior to calling `IBaseBot.go()`.

If this method is called multiple times, the last call before `IBaseBot.go()` is executed, counts.

This method will cancel the effect of prior calls to `IBaseBot.setTargetSpeed(double)` as the `IBot.setForward(double)` and setBack methods calls the `IBaseBot.setTargetSpeed(double)` for each turn until `IBot.getDistanceRemaining()` reaches 0.

Specified by:

`setBack` in interface `IBot`

Parameters:

`distance` - is the distance to move backward. If negative, the bot will move forward. If `Double.POSITIVE_INFINITY` the bot will move backward infinitely. If `Double.NEGATIVE_INFINITY` the bot will move forward infinitely.

See Also:

`IBot.back(double)`, `IBot.setForward(double)`, `IBot.forward(double)`, `IBot.getDistanceRemaining()`, `IBaseBot.setTargetSpeed(double)`

back

```
public final void back(double distance)
```

Moves the bot backward until it has traveled a specific distance from its current position, or it is moving into an obstacle. The speed is limited by `IBaseBot.setMaxSpeed(double)`.

When the bot is moving forward, the `Constants.ACCELERATION` determine the acceleration of the bot that adds 1 additional unit to the speed per turn while accelerating. However, the bot is faster at braking. The `Constants.DECELERATION` determine the deceleration of the bot that subtracts 2 units from the speed per turn.

This call is executed immediately by calling `IBaseBot.go()` in the code behind. This method will block until it has been completed, which can take one to several turns. New commands will first take place after this method is completed. If you need to execute multiple commands in parallel, use **setter** methods instead of this blocking method.

This method will cancel the effect of prior calls to `IBaseBot.setTargetSpeed(double)`, `IBot.setForward(double)`, and `IBot.setBack(double)` methods.

Specified by:

`back` in interface `IBot`

See Also:

`IBot.setForward(double)`, `IBot.setBack(double)`, `IBot.forward(double)`, `IBot.getDistanceRemaining()`, `IBaseBot.setTargetSpeed(double)`

getDistanceRemaining

```
public final double getDistanceRemaining()
```

Returns the distance remaining till the bot has finished moving after having called `IBot.setForward(double)`, `IBot.setBack(double)`, `IBot.forward(double)`, or `IBot.back(double)`. When the distance remaining has reached 0, the bot has finished its current move.

When the distance remaining is positive, the bot is moving forward. When the distance remaining is negative, the bot is moving backward.

Specified by:

`getDistanceRemaining` in interface `IBot`

Returns:

The remaining distance to move before its current movement is completed. If `Double.POSITIVE_INFINITY` the bot will move forward infinitely. If `Double.NEGATIVE_INFINITY` the bot will move backward infinitely.

See Also:

`IBot.setForward(double)`, `IBot.setBack(double)`, `IBot.forward(double)`, `IBot.back(double)`

setTurnLeft

```
public final void setTurnLeft(double degrees)
```

Set the bot to turn to the left (following the increasing degrees of the unit circle) until it turned the specified amount of degrees. That is, when `IBot.getTurnRemaining()` is 0. The amount of degrees to turn each turn is limited by `IBaseBot.setMaxTurnRate(double)`.

This method will first be executed when `IBaseBot.go()` is called making it possible to call other set methods after execution. This makes it possible to set the bot to move, turn the body, radar, gun, and also fire the gun in parallel in a single turn when calling `IBaseBot.go()`. But notice that this is only possible to execute multiple methods in parallel by using **setter** methods only prior to calling `IBaseBot.go()`.

`setTurnLeft` in interface `IBot`

Parameters:

`degrees` - is the amount of degrees to turn left. If negative, the bot will turn right. If `Double.POSITIVE_INFINITY` the bot will turn left infinitely. If `Double.NEGATIVE_INFINITY` the bot will turn right infinitely.

See Also:

Unit circle, `IBot.setTurnRight(double)`, `IBot.turnRight(double)`, `IBot.turnLeft(double)`, `IBot.getTurnRemaining()`, `IBaseBot.setTurnRate(double)`

turnLeft

```
public final void turnLeft(double degrees)
```

Turn the bot to the left (following the increasing degrees of the unit circle) until it turned the specified amount of degrees. That is, when `IBot.getTurnRemaining()` is 0. The amount of degrees to turn each turn is limited by `IBaseBot.setMaxTurnRate(double)`.

This call is executed immediately by calling `IBaseBot.go()` in the code behind. This method will block until it has been completed, which can take one to several turns. New commands will first take place after this method is completed. If you need to execute multiple commands in parallel, use **setter** methods instead of this blocking method.

This method will cancel the effect of prior calls to `IBot.setTurnLeft(double)` and `IBot.setTurnRight(double)`.

Specified by:

`turnLeft` in interface `IBot`

Parameters:

`degrees` - is the amount of degrees to turn left. If negative, the bot will turn right. If `Double.POSITIVE_INFINITY` the bot will turn left infinitely. If `Double.NEGATIVE_INFINITY` the bot will turn right infinitely.

See Also:

Unit circle, `IBot.setTurnLeft(double)`, `IBot.setTurnRight(double)`, `IBot.turnRight(double)`, `IBot.getTurnRemaining()`, `IBaseBot.setTurnRate(double)`

setTurnRight

```
public final void setTurnRight(double degrees)
```

other set methods after execution. This makes it possible to set the bot to move, turn the body, radar, gun, and also fire the gun in parallel in a single turn when calling `IBaseBot.go()`. But notice that this is only possible to execute multiple methods in parallel by using **setter** methods only prior to calling `IBaseBot.go()`.

If this method is called multiple times, the last call before `IBaseBot.go()` is executed, counts.

This method will cancel the effect of prior calls to `IBot.setTurnLeft(double)`.

Specified by:

`setTurnRight` in interface `IBot`

Parameters:

`degrees` - is the amount of degrees to turn right. If negative, the bot will turn left. If `Double.POSITIVE_INFINITY` the bot will turn right infinitely. If `Double.NEGATIVE_INFINITY` the bot will turn left infinitely.

See Also:

`IBot.setTurnLeft(double)`, `IBot.turnRight(double)`, `IBot.turnLeft(double)`, `IBot.getTurnRemaining()`, `IBaseBot.setTurnRate(double)`

turnRight

```
public final void turnRight(double degrees)
```

Turn the bot to the right (following the increasing degrees of the unit circle) until it turned the specified amount of degrees. That is, when `IBot.getTurnRemaining()` is 0. The amount of degrees to turn each turn is limited by `IBaseBot.setMaxTurnRate(double)`.

This call is executed immediately by calling `IBaseBot.go()` in the code behind. This method will block until it has been completed, which can take one to several turns. New commands will first take place after this method is completed. If you need to execute multiple commands in parallel, use **setter** methods instead of this blocking method.

This method will cancel the effect of prior calls to `IBot.setTurnLeft(double)` and `IBot.setTurnRight(double)`.

Specified by:

`turnRight` in interface `IBot`

Parameters:

`degrees` - is the amount of degrees to turn right. If negative, the bot will turn left. If `Double.POSITIVE_INFINITY` the bot will turn right infinitely. If `Double.NEGATIVE_INFINITY` the bot will turn left infinitely.

getTurnRemaining

```
public final double getTurnRemaining()
```

Returns the remaining turn in degrees till the bot has finished turning after having called `IBot.setTurnLeft(double)`, `IBot.setTurnRight(double)`, `IBot.turnLeft(double)`, or `IBot.turnRight(double)`. When the turn remaining has reached 0, the bot has finished turning.

When the turn remaining is positive, the bot is turning to the left (along the unit circle). When the turn remaining is negative, the bot is turning to the right.

Specified by:

`getTurnRemaining` in interface `IBot`

Returns:

The remaining degrees to turn before its current turning is completed. If `Double.POSITIVE_INFINITY` the bot will turn left infinitely. If `Double.NEGATIVE_INFINITY` the bot will turn right infinitely.

See Also:

`IBot.setTurnLeft(double)`, `IBot.setTurnRight(double)`, `IBot.turnLeft(double)`, `IBot.turnRight(double)`

setTurnGunLeft

```
public final void setTurnGunLeft(double degrees)
```

Set the gun to turn to the left (following the increasing degrees of the unit circle) until it turned the specified amount of degrees. That is, when `IBot.getGunTurnRemaining()` is 0. The amount of degrees to turn each turn is limited by `IBaseBot.setMaxGunTurnRate(double)`.

This method will first be executed when `IBaseBot.go()` is called making it possible to call other set methods after execution. This makes it possible to set the bot to move, turn the body, radar, gun, and also fire the gun in parallel in a single turn when calling `IBaseBot.go()`. But notice that this is only possible to execute multiple methods in parallel by using **setter** methods only prior to calling `IBaseBot.go()`.

If this method is called multiple times, the last call before `IBaseBot.go()` is executed, counts.

This method will cancel the effect of prior calls to `IBot.setTurnGunRight(double)`.

`Double.POSITIVE_INFINITY` the gun will turn left infinitely. If `Double.NEGATIVE_INFINITY` the gun will turn right infinitely.

See Also:

Unit circle, `IBot.setTurnGunRight(double)`, `IBot.turnGunRight(double)`, `IBot.turnGunLeft(double)`, `IBot.getGunTurnRemaining()`, `IBaseBot.setGunTurnRate(double)`

turnGunLeft

```
public final void turnGunLeft(double degrees)
```

Turn the gun to the left (following the increasing degrees of the unit circle) until it turned the specified amount of degrees. That is, when `IBot.getGunTurnRemaining()` is 0. The amount of degrees to turn each turn is limited by `IBaseBot.setMaxGunTurnRate(double)`.

This call is executed immediately by calling `IBaseBot.go()` in the code behind. This method will block until it has been completed, which can take one to several turns. New commands will first take place after this method is completed. If you need to execute multiple commands in parallel, use **setter** methods instead of this blocking method.

This method will cancel the effect of prior calls to `IBot.setTurnGunLeft(double)` and `IBot.setTurnGunRight(double)`.

Specified by:

`turnGunLeft` in interface `IBot`

Parameters:

`degrees` - is the amount of degrees to turn left. If negative, the gun will turn right. If `Double.POSITIVE_INFINITY` the gun will turn left infinitely. If `Double.NEGATIVE_INFINITY` the gun will turn right infinitely.

See Also:

Unit circle, `IBot.setTurnGunLeft(double)`, `IBot.setTurnGunRight(double)`, `IBot.turnGunRight(double)`, `IBot.getGunTurnRemaining()`, `IBaseBot.setGunTurnRate(double)`

setTurnGunRight

```
public final void setTurnGunRight(double degrees)
```

Set the gun to turn to the right (following the decreasing degrees of the unit circle) until it turned the specified amount of degrees. That is, when `IBot.getGunTurnRemaining()` is 0.

body, radar, gun, and also fire the gun in parallel in a single turn when calling `IBaseBot.go()`. But notice that this is only possible to execute multiple methods in parallel by using **setter** methods only prior to calling `IBaseBot.go()`.

If this method is called multiple times, the last call before `IBaseBot.go()` is executed, counts.

This method will cancel the effect of prior calls to `IBot.setTurnGunLeft(double)`.

Specified by:

`setTurnGunRight` in interface `IBot`

Parameters:

`degrees` - is the amount of degrees to turn right. If negative, the gun will turn left. If `Double.POSITIVE_INFINITY` the gun will turn right infinitely. If `Double.NEGATIVE_INFINITY` the gun will turn left infinitely.

See Also:

Unit circle, `IBot.setTurnGunLeft(double)`, `IBot.turnGunRight(double)`, `IBot.turnGunLeft(double)`, `IBot.getGunTurnRemaining()`, `IBaseBot.setGunTurnRate(double)`

turnGunRight

```
public final void turnGunRight(double degrees)
```

Turn the gun to the right (following the decreasing degrees of the unit circle) until it turned the specified amount of degrees. That is, when `IBot.getGunTurnRemaining()` is 0. The amount of degrees to turn each turn is limited by `IBaseBot.setMaxGunTurnRate(double)`.

This call is executed immediately by calling `IBaseBot.go()` in the code behind. This method will block until it has been completed, which can take one to several turns. New commands will first take place after this method is completed. If you need to execute multiple commands in parallel, use **setter** methods instead of this blocking method.

This method will cancel the effect of prior calls to `IBot.setTurnGunLeft(double)` and `IBot.setTurnGunRight(double)`.

Specified by:

`turnGunRight` in interface `IBot`

Parameters:

`degrees` - is the amount of degrees to turn right. If negative, the gun will turn left. If `Double.POSITIVE_INFINITY` the gun will turn right infinitely. If `Double.NEGATIVE_INFINITY` the gun will turn left infinitely.

getGunTurnRemaining

```
public final double getGunTurnRemaining()
```

Returns the remaining turn in degrees till the gun has finished turning after having called `IBot.setTurnGunLeft(double)`, `IBot.setTurnGunRight(double)`, `IBot.turnGunLeft(double)`, or `IBot.turnGunRight(double)`. When the turn remaining has reached 0, the gun has finished turning.

When the turn remaining is positive, the bot is turning to the left (along the unit circle). When the turn remaining is negative, the bot is turning to the right.

Specified by:

`getGunTurnRemaining` in interface `IBot`

Returns:

The remaining degrees to turn the gun before its current turning is completed. If `Double.POSITIVE_INFINITY` the gun will turn left infinitely. If `Double.NEGATIVE_INFINITY` the gun will turn right infinitely.

See Also:

`IBot.setTurnGunLeft(double)`, `IBot.setTurnGunRight(double)`, `IBot.turnGunLeft(double)`, `IBot.turnGunRight(double)`

setTurnRadarLeft

```
public final void setTurnRadarLeft(double degrees)
```

Set the radar to turn to the left (following the increasing degrees of the unit circle) until it turned the specified amount of degrees. That is, when `IBot.getRadarTurnRemaining()` is 0. The amount of degrees to turn each turn is limited by `IBaseBot.setMaxRadarTurnRate(double)`.

This method will first be executed when `IBaseBot.go()` is called making it possible to call other set methods after execution. This makes it possible to set the bot to move, turn the body, radar, gun, and also fire the gun in parallel in a single turn when calling `IBaseBot.go()`. But notice that this is only possible to execute multiple methods in parallel by using **setter** methods only prior to calling `IBaseBot.go()`.

If this method is called multiple times, the last call before `IBaseBot.go()` is executed, counts.

This method will cancel the effect of prior calls to `IBot.setTurnRadarRight(double)`.

`Double.POSITIVE_INFINITY` the radar will turn left infinitely. If `Double.NEGATIVE_INFINITY` the radar will turn right infinitely.

See Also:

Unit circle, `IBot.setTurnRadarRight(double)`, `IBot.turnRadarRight(double)`, `IBot.turnRadarLeft(double)`, `IBot.getRadarTurnRemaining()`, `IBaseBot.setRadarTurnRate(double)`

turnRadarLeft

```
public final void turnRadarLeft(double degrees)
```

Turn the radar to the left (following the increasing degrees of the unit circle) until it turned the specified amount of degrees. That is, when `IBot.getRadarTurnRemaining()` is 0. The amount of degrees to turn each turn is limited by `IBaseBot.setMaxRadarTurnRate(double)`.

This call is executed immediately by calling `IBaseBot.go()` in the code behind. This method will block until it has been completed, which can take one to several turns. New commands will first take place after this method is completed. If you need to execute multiple commands in parallel, use **setter** methods instead of this blocking method.

This method will cancel the effect of prior calls to `IBot.setTurnRadarLeft(double)` and `IBot.setTurnRadarRight(double)`.

Specified by:

`turnRadarLeft` in interface `IBot`

Parameters:

`degrees` - is the amount of degrees to turn left. If negative, the radar will turn right. If `Double.POSITIVE_INFINITY` the radar will turn left infinitely. If `Double.NEGATIVE_INFINITY` the radar will turn right infinitely.

See Also:

Unit circle, `IBot.setTurnRadarLeft(double)`, `IBot.setTurnRadarRight(double)`, `IBot.turnRadarRight(double)`, `IBot.getRadarTurnRemaining()`, `IBaseBot.setRadarTurnRate(double)`

setTurnRadarRight

```
public final void setTurnRadarRight(double degrees)
```

This method will first be executed when `IBaseBot.go()` is called making it possible to call other set methods after execution. This makes it possible to set the bot to move, turn the body, radar, gun, and also fire the gun in parallel in a single turn when calling `IBaseBot.go()`. But notice that this is only possible to execute multiple methods in parallel by using **setter** methods only prior to calling `IBaseBot.go()`.

If this method is called multiple times, the last call before `IBaseBot.go()` is executed, counts.

This method will cancel the effect of prior calls to `IBot.setTurnRadarLeft(double)` and setTurnRadarRight(double).

Specified by:

`setTurnRadarRight` in interface `IBot`

Parameters:

`degrees` - is the amount of degrees to turn right. If negative, the radar will turn left. If `Double.POSITIVE_INFINITY` the radar will turn right infinitely. If `Double.NEGATIVE_INFINITY` the radar will turn left infinitely.

See Also:

Unit circle, `IBot.setTurnRadarLeft(double)`, `IBot.turnRadarLeft(double)`, `IBot.turnRadarRight(double)`, `IBot.getRadarTurnRemaining()`, `IBaseBot.setRadarTurnRate(double)`

turnRadarRight

```
public final void turnRadarRight(double degrees)
```

Turn the radar to the right (following the increasing degrees of the unit circle) until it turned the specified amount of degrees. That is, when `IBot.getRadarTurnRemaining()` is 0. The amount of degrees to turn each turn is limited by `IBaseBot.setMaxRadarTurnRate(double)`.

This call is executed immediately by calling `IBaseBot.go()` in the code behind. This method will block until it has been completed, which can take one to several turns. New commands will first take place after this method is completed. If you need to execute multiple commands in parallel, use **setter** methods instead of this blocking method.

This method will cancel the effect of prior calls to `IBot.setTurnRadarLeft(double)` and `IBot.setTurnRadarRight(double)`.

Specified by:

`turnRadarRight` in interface `IBot`

Parameters:

Unit circle, IBot.setTurnRadarLeft(double), IBot.setTurnRadarRight(double), IBot.turnRadarRight(double), IBot.getRadarTurnRemaining(), IBaseBot.setRadarTurnRate(double)

getRadarTurnRemaining

```
public final double getRadarTurnRemaining()
```

Returns the remaining turn in degrees till the radar has finished turning after having called `IBot.setTurnRadarLeft(double)`, `IBot.setTurnRadarRight(double)`, `IBot.turnRadarLeft(double)`, or `IBot.turnRadarRight(double)`. When the turn remaining has reached 0, the radar has finished turning.

When the turn remaining is positive, the bot is turning to the left (along the unit circle). When the turn remaining is negative, the bot is turning to the right.

Specified by:

`getRadarTurnRemaining` in interface `IBot`

Returns:

The remaining degrees to turn the radar before its current turning is completed. If `Double.POSITIVE_INFINITY` the radar will turn left infinitely. If `Double.NEGATIVE_INFINITY` the radar will turn right infinitely.

See Also:

`IBot.setTurnRadarLeft(double)`, `IBot.setTurnRadarRight(double)`, `IBot.turnRadarLeft(double)`, `IBot.turnRadarRight(double)`

fire

```
public final void fire(double firepower)
```

Fire the gun in the direction as the gun is pointing.

Note that your bot is spending energy when firing a bullet, the amount of energy used for firing the bullet is taken from the bot. The amount of energy loss is equal to firepower.

If the bullet hits an opponent bot, you will gain energy from the bullet hit. When hitting another bot, your bot will be rewarded and retrieve an energy boost of 3x firepower.

The gun will only fire when the firepower is at `Constants.MIN_FIREPOWER` or higher. If the firepower is more than `Constants.MAX_FIREPOWER`, the power will be truncated to the max firepower.

The amount of energy used for firing the gun is subtracted from the bots total energy. The amount of damage dealt by a bullet hitting another bot is 4x firepower, and if the firepower is greater than 1 it will do an additional 2 x (firepower - 1) damage.

The firepower is truncated to `Constants.MIN_FIREPOWER` and `Constants.MAX_FIREPOWER` if the firepower exceeds these values.

This call is executed immediately by calling `IBaseBot.go()` in the code behind. This method will block until it has been completed, which can take one to several turns. New commands will first take place after this method is completed. If you need to execute multiple commands in parallel, use **setter** methods instead of this blocking method.

This method will cancel the effect of prior calls to `IBaseBot.setFire(double)`.

Specified by:

`fire` in interface `IBot`

Parameters:

`firepower` - is the amount of energy spent on firing the gun. You cannot spend more energy than available from the bot. The bullet power must be greater than `Constants.MIN_FIREPOWER`.

See Also:

`IBaseBot.onBulletFired(dev.robocode.tankroyale.botapi.events.BulletFiredEvent)`, `IBaseBot.setFire(double)`, `IBaseBot.getGunHeat()`, `IBaseBot.getGunCoolingRate()`

stop

```
public final void stop()
```

Stop all movement including turning the gun and radar. The remaining movement is saved for a call to `IBaseBot.setResume()` or `IBot.resume()`. This method has no effect, if it has already been called.

This call is executed immediately by calling `IBaseBot.go()` in the code behind. This method will block until it has been completed, which can take one to several turns. New commands will first take place after this method is completed. If you need to execute multiple commands in parallel, use **setter** methods instead of this blocking method.

Specified by:

`stop` in interface `IBot`

See Also:

`IBot.resume()`, `IBaseBot.setResume()`, `IBaseBot.setStop()`

Stop all movement including turning the gun and radar. The remaining movement is saved for a call to `IBaseBot.setResume()` or `IBot.resume()`.

This call is executed immediately by calling `IBaseBot.go()` in the code behind. This method will block until it has been completed, which can take one to several turns. New commands will first take place after this method is completed. If you need to execute multiple commands in parallel, use **setter** methods instead of this blocking method.

Specified by:

`stop` in interface `IBot`

Parameters:

`overwrite` - is set to `true` if the movement saved by a previous call to this method or `IBaseBot.setStop()` must be overridden with the current movement. When set to `false` this method is identical to `IBaseBot.setStop()`.

See Also:

`IBot.resume()`, `IBaseBot.setResume()`, `IBaseBot.setStop()`

resume

```
public final void resume()
```

Resume the movement prior to calling the `IBaseBot.setStop()` or `IBot.stop()` method. This method has no effect, if it has already been called.

This call is executed immediately by calling `IBaseBot.go()` in the code behind. This method will block until it has been completed, which can take one to several turns. New commands will first take place after this method is completed. If you need to execute multiple commands in parallel, use **setter** methods instead of this blocking method.

Specified by:

`resume` in interface `IBot`

See Also:

`IBot.stop()`, `IBaseBot.setStop()`, `IBaseBot.setResume()`

rescan

```
public final void rescan()
```

Specified by:

`rescan` in interface `IBot`

See Also:

`IBot.stop()`

waitFor

```
public final void waitFor(Condition condition)
```

Blocks until a condition is met, i.e. when a `Condition.test()` returns true.

Specified by:

`waitFor` in interface `IBot`

Parameters:

`condition` - is the condition that must be met before this method will stop waiting.

See Also:

`Condition`,
`IBaseBot.onCustomEvent(dev.robocode.tankroyale.botapi.events.CustomEvent)`