Package hockey.api

Welcome to the contest, please read the description!

See:
Description

Interface Summary	
IGoalKeeper	An `IObject` representing a goal-keeper on the ice.
IGoalKeeperControl	An interface to goal-keeper control.
IObject	An object on the ice.
IPlayer	An `IObject` representing a player on the ice.
IPlayerControl	An interface to player control.
IPuck	An `IObject` representing a puck on the ice.
ITeam	Your team should implement this interface!

Class Summary	
GoalKeeper	The goal-keeper in your team should extend this class!
Player	Every player in your team should extend this class!
Position	An `IObject` implementation that represents a position on the ice.
Util	The class `Util` contains methods for performing basic numeric operations such as the elementary squaring, euclidean distance, angle conversion, trigonometric functions in degrees and clamping.

Package hockey.api Description

Welcome to the contest, please read the description!

The Game

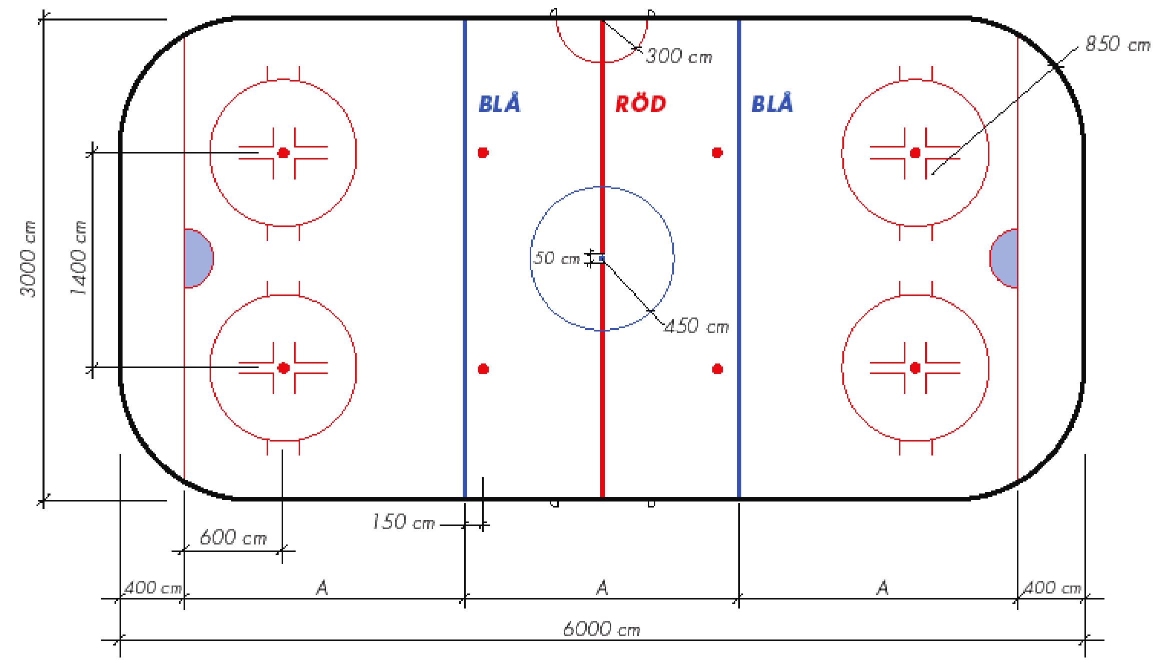
The game is hockey, the objective is to score...

Cheating is forbidden. Cheating, or attempts to cheat, leads to disqualification.

Note! You may NOT use `java.lang.System`.

Rink measurements, coordinate system and units

Official measurements of the rink:



- 3A is 5200 cm, so A is approximately 1733 cm, but 3A/2 is exactly 2600 cm (the distance from the centre line to the goal line).
- Goals are about 180 cm wide, but the goal area's diameter is 360 cm.

The game's coordinate system has its origin in the centre spot on the ice.

- The x-axis is along the long side of the rink, positive towards your opponent's goal.
- The y-axis is along the short side of the rink, positive to your goal-keeper's left.
- Angles are relative to the x-axis, positive towards the y-axis.

Basic units of the game are:

- cm - distances and coordinates in centimetres.
- s - time in seconds.
- degrees - angles in degrees.

Derived from these are units for speed, acceleration and turn speed:

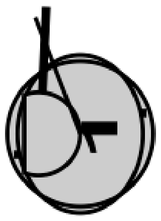
- cm/s - speed in centimetres per second.
- cm/s^2 - acceleration in centimetres per second square.
- degrees/s - turn speed in degrees per second.

Short approximate conversion table for speed:

cm/s	km/h	m/s
444	16	4.44
1000	36	10.00
1111	40	11.11
2777	100	27.77
4444	160	44.44

Player control and specifications

All moving objects on the ice are modelled as circles, even if they are not drawn like that. The stick is controlled relative to the player's centre, but is drawn from the front of the player to a position a bit behind and further away from where the puck is held.



Physical specifications

There are four types of solid objects in the game:

- **Players** (with one exception, see below).

Property	Value (regular player/goalkeeper)
Width (cm)	70/80
Mass (kg)	75/150 (yes, he's a steady fellow!)
Acceleration (cm/s^2)	444
Angular acceleration (deg/s^2)	720

  Some of these, and several more properties of a player can be found in the Player API.

  Note!! Due to the mysterious magical energies flowing from somewhere near your goalcage, your players (except the goalkeeper in his special magic armour) are not solid with respect to the puck within your own goalzone.

- **The puck**

Property	Value
Width (cm)	7.5
Mass (kg)	0.170

- **The goal cages** Players, as well as the puck, bounce against the interior as well as the exterior of the goal cages.

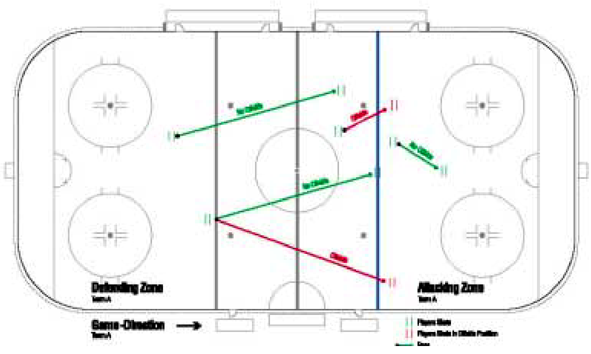
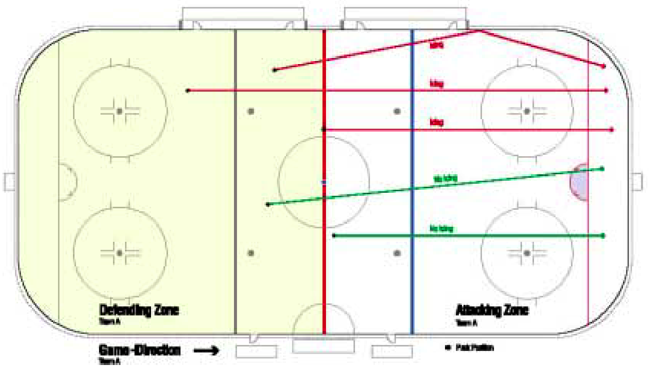
- **The rink** Players, as well as the puck, bounce against the rink.

Collisions between these solid objects occur more or less according to the laws of physics. As mentioned above, the players (and of course the puck) are modelled as circles.

Rules

The folowing rules are judged by the game:

- **Offside** - an attacking player may not arrive before the puck into the attack zone (over the far blue line).
- **Icing** - the puck may not shoot the puck from his own half of the rink past his opponent's goal line, except if he scores or the puck passes through the goal area.
- **Blocking** - the puck may not be immobile for more than a couple of seconds .

Offside	Icing
	 Icing Icing Icing No Icing No Icing Defending Zone Attacking Zone Game-Direction

A strong recommendation

(from our experiences of similar contests)

Plan realistically! Five hours is a very short time for the task. Implement something simple that works, to get a feel for the system, then proceed with your masterplan. It's much more fun watching a working team afterwards!

Disclaimer

Although this game tries to emulate a fast and simplified version of real hockey, it is in no way guaranteed to be faithful to reality. *In all cases the game's decisions are valid.* Your objective is to write a good team *in this game, as it is*.

Good Luck!

Now read through the rest of the API! :)

Constant Field Values

Contents

hockey.api.*

hockey.api.IGoalKeeperControl		
public static final int	MAX_GLIDE	444
public static final int	MAX_THROW_SPEED	1111

hockey.api.IPlayerControl		
public static final int	ACCELERATION	444
public static final int	MAX_SHOT_SPEED	4444
public static final int	MAX_SPEED	1111
public static final int	MAX_STICK_ANGLE	135
public static final int	MAX_STICK_R	70
public static final int	MAX_TURN_SPEED	180
public static final int	MIN_STICK_ANGLE	0
public static final int	MIN_STICK_R	40

hockey.api

Class Util

```
java.lang.Object
  |
  +--hockey.api.Util
```

public class **Util**
extends java.lang.Object

The class `Util` contains methods for performing basic numeric operations such as the elementary squaring, euclidean distance, angle conversion, trigonometric functions in degrees and clamping.

Many of the `Util` functions simply delegate to the equivalent functions in `Math` for their implementations.

See Also:
`Math`

Constructor Summary

`Util()`

Method Summary

`static double`	`clamp(double a, double x, double b)` Clamps a `double` value to a given range.
`static int`	`clamp(int a, int x, int b)` Clamps an `int` value to a given range.
`static double`	`clampAbs(double x, double a)` Clamps a `double` value between zero and a given maximum value.
`static int`	`clampAbs(int x, int a)` Clamps an `int` value to a given maximum absolute value.
`static double`	`clampPos(double x, double b)` Clamps a `double` value between zero and a given maximum value.
`static int`	`clampPos(int x, int b)` Clamps an `int` value between zero and a given maximum value.
`static double`	`cosd(double deg)` Returns the trigonometric cosine of an angle given in degrees.
`static double`	`dangle(double v1, double v2)` Returns the minimal angular distance between two angles in degrees.
`static double`	`datan2(double y, double x)` Returns the angle in degrees to a point (x,y).
`static double`	`datan2(IObject to, IObject origin)` Returns the absolute angle in degrees to an `IObject` with another `IObject` as origin.
`static double`	`deg(double rad)` Converts an angle measured in radians to the equivalent angle measured in degrees.
`static double`	`dist(double x, double y)` Returns the distance from the origin to a coordinate.
`static double`	`dist(IObject o1, IObject o2)` Returns the distance between two `IObject`s.
`static double`	`dist2(double x, double y)` Returns the squared distance from the origin to a `double` coordinate.
`static int`	`dist2(int x, int y)` Returns the squared distance from the origin to an `int` coordinate.
`static int`	

	dist2(IObject o1, IObject o2) Returns the squared distance between two `IObject`s.
static double	**rad**(double deg) Converts an angle measured in degrees to the equivalent angle measured in radians.
static double	**sind**(double deg) Returns the trigonometric sine of an angle given in degrees.
static double	**sqr**(double x) Returns the square of a `double` value.
static int	**sqr**(int x) Returns the square of an `int` value.
static double	**tand**(double deg) Returns the trigonometric tangent of an angle given in degrees.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Util

```
public Util()
```

Method Detail

sqr

```
public static int sqr(int x)
```

Returns the square of an `int` value.

Parameters:
`x` - an `int` value.
Returns:
the square of `x`.

---

sqr

```
public static double sqr(double x)
```

Returns the square of a `double` value.

Parameters:
`x` - a `double` value.
Returns:
the square of `x`.

---

dist

```
public static double dist(double x,
                          double y)
```

Returns the distance from the origin to a coordinate.

Parameters:
`x` - the x-coordinate.
`y` - the y-coordinate.
Returns:
the distance from the origin to (x, y).

---

dist

```
public static double dist(IObject o1,
                          IObject o2)
```

Returns the distance between two `IObject`s.

Parameters:
`o1` - the first `IObject`.
Returns:
the distance between the two `IObject`s.

---

dist2

```
public static double dist2(double x,
                           double y)
```

Returns the squared distance from the origin to a `double` coordinate.

Parameters:
`x` - the x-coordinate.
`y` - the y-coordinate.
Returns:
the squared distance from the origin to (x, y).

---

dist2

```
public static int dist2(int x,
                        int y)
```

Returns the squared distance from the origin to an `int` coordinate.

Parameters:
`x` - the x-coordinate.
`y` - the y-coordinate.
Returns:
the squared distance from the origin to (x, y).

---

dist2

```
public static int dist2(IObject o1,
                        IObject o2)
```

Returns the squared distance between two `IObject`s.

Parameters:
`o1` - the first `IObject`.
Returns:
the distance between the two `IObject`s.

---

dangle

```
public static double dangle(double v1,
                            double v2)
```

Returns the minimal angular distance between two angles in degrees. Return values are in the range -180 to 180. This method is the same as calling `Math.IEEEremainder(v1 - v2, 360)`.

Parameters:
`v1` - the first angle, in degrees.
`v2` - the second angle, in degrees.
Returns:
the minimal angular distance between the two angles, in degrees.

---

rad

```
public static double rad(double deg)
```

Converts an angle measured in degrees to the equivalent angle measured in radians. This method is the same as calling `Math.toRadians(deg)`.

Parameters:
`deg` - an angle, in degrees.
Returns:
the measurement of the angle `deg` in radians.

---

deg

```
public static double deg(double rad)
```

Converts an angle measured in radians to the equivalent angle measured in degrees. This method is the same as calling `Math.toDegrees(rad)`.

Parameters:
`rad` - an angle, in radians.
Returns:
the measurement of the angle `rad` in degrees.

---

cosd

```
public static double cosd(double deg)
```

Returns the trigonometric cosine of an angle given in degrees. This method is the same as calling `Math.cos(rad(deg))`.

Parameters:
`deg` - the angle, in degrees.
Returns:
the cosine of the argument.

---

sind

```
public static double sind(double deg)
```

Returns the trigonometric sine of an angle given in degrees. This method is the same as calling `Math.sin(rad(deg))`.

Parameters:
`deg` - the angle, in degrees.
Returns:
the sine of the argument.

---

tand

```
public static double tand(double deg)
```

Returns the trigonometric tangent of an angle given in degrees. This method is the same as calling `Math.tan(rad(deg))`.

Parameters:
`deg` - the angle, in degrees.
Returns:
the tangent of the argument.

---

datan2

```
public static double datan2(double y,
                            double x)
```

Returns the angle in degrees to a point (x,y). This method is the same as calling `deg(Math.atan2(y, x))`. Note that the point is given with the y-coordinate first, as for `Math.atan2`.

Parameters:

`y` - the y-coordinate of the point.
`x` - the x-coordinate of the point.

Returns:

the angle from the origin to the point, in degrees.

---

datan2

```
public static double datan2(IObject to,
                            IObject origin)
```

Returns the absolute angle in degrees to an `IObject` with another `IObject` as origin. The angle is absolute in the x-y-coordinate system of the `IObject`s, the heading of the origin `IObject` is not considered.

Parameters:

`to` - the `IObject` to get the absolute angle to.
`origin` - the `IObject` to use as origin.

Returns:

the absolute angle to `to` with `origin` as origin.

---

clamp

```
public static int clamp(int a,
                        int x,
                        int b)
```

Clamps an `int` value to a given range. The returned value is `a` if `x<a`, `x` if it is in the range, or `b` if `x>b`. The metod is the same as calling `Math.min(Math.max(a, x), b)`.

Parameters:

`a` - the minimum value of the range.
`x` - the `int` value.
`b` - the maximum value of the range.

Returns:

the value `x` clamped to the range `a`-`b`.

---

clampPos

```
public static int clampPos(int x,
                           int b)
```

Clamps an `int` value between zero and a given maximum value. The returned value is `0` if `x<0`, `x` if it is in the range, or `b` if `x>b`. The metod is the same as calling `clamp(0, x, b)`.

Parameters:

`x` - the `int` value.
`b` - the maximum value of the range.

Returns:

the value `x` clamped to the range `0`-`b`.

---

clampAbs

```
public static int clampAbs(int x,
                           int a)
```

Clamps an `int` value to a given maximum absolute value. The returned value is `-a` if `x<-a`, `x` if it is in the range, or `a` if `x>a`. The metod is the same as calling `clamp(-a, x, a)`.

Parameters:

`x` - the `int` value.
`a` - the maximum absolute value.

Returns:

the value `x` clamped to the range `-a`-`a`.

---

clamp

```
public static double clamp(double a,
                           double x,
                           double b)
```

Clamps a `double` value to a given range. The returned value is `a` if `x<a`, `x` if it is in the range, or `b` if `x>b`. The metod is the same as calling `Math.min(Math.max(a, x), b)`.

Parameters:
`a` - the minimum value of the range.
`x` - the `int` value.
`b` - the maximum value of the range.

Returns:
the value `x` clamped to the range `a`-`b`.

---

clampPos

```
public static double clampPos(double x,
                              double b)
```

Clamps a `double` value between zero and a given maximum value. The returned value is `0` if `x<0`, `x` if it is in the range, or `b` if `x>b`. The metod is the same as calling `clamp(0, x, b)`.

Parameters:
`x` - the `int` value.
`b` - the maximum value of the range.

Returns:
the value `x` clamped to the range `0`-`b`.

---

clampAbs

```
public static double clampAbs(double x,
                              double a)
```

Clamps a `double` value between zero and a given maximum value. The returned value is `0` if `x<0`, `x` if it is in the range, or `a` if `x>a`. The metod is the same as calling `clamp(-a, x, a)`.

Parameters:
`x` - the `int` value.
`a` - the maximum absolute value.

Returns:
the value `x` clamped to the range `-a`-`a`.

---

hockey.api

Interface IObject

All Known Subinterfaces:
IGoalKeeper, IPlayer, IPuck

All Known Implementing Classes:
GoalKeeper, Player, Position

public interface **IObject**

An object on the ice. Every object in the game has at least a position, heading and speed.

Method Summary

`int`	`getHeading()` Returns the absolute heading in degrees.
`int`	`getSpeed()` Returns the speed in cm/s.
`int`	`getX()` Returns the x-coordinate in cm.
`int`	`getY()` Returns the y-coordinate in cm.

Method Detail

getX

```
public int getX()
```

Returns the x-coordinate in cm.

Returns:
the x-coordinate, in cm.

getY

```
public int getY()
```

Returns the y-coordinate in cm.

Returns:
the y-coordinate, in cm.

getHeading

```
public int getHeading()
```

Returns the absolute heading in degrees.

Returns:
the absolute heading, in degrees.

getSpeed

```
public int getSpeed()
```

Returns the speed in cm/s.

Returns:
the speed, in cm/s.

---

hockey.api

Interface IPuck

All Superinterfaces:
IObject

public interface **IPuck**
extends IObject

An `IObject` representing a puck on the ice.

Method Summary

`IPlayer`	**`getHolder`**`()` Returns the holder of the puck, or `null` if the puck is not held.
`boolean`	**`isHeld`**`()` Returns whether the puck is held by any player.

Methods inherited from interface hockey.api.IObject
`getHeading, getSpeed, getX, getY`

Method Detail

isHeld

```
public boolean isHeld()
```

Returns whether the puck is held by any player.

Returns:
`true` if the puck is held.

getHolder

```
public IPlayer getHolder()
```

Returns the holder of the puck, or `null` if the puck is not held.

Returns:
the holder of the puck, or `null` if not held.

hockey.api

Interface IPlayer

All Superinterfaces:
IObject

All Known Subinterfaces:
IGoalKeeper

All Known Implementing Classes:
GoalKeeper, Player

public interface **IPlayer**
extends IObject

An `IObject` representing a player on the ice.

See Also:
`IObject`

Method Summary

`int`	**`getIndex`**`()` Returns this player's index.
`IObject`	**`getStick`**`()` Returns the current position of the player's stick, as an `IObject`.
`int`	**`getStickAngle`**`()` Returns the stick relative angle relative to the player's heading, in degrees.
`int`	**`getStickR`**`()` Returns the stick distace from the player's centre, in cm.
`int`	**`getStickX`**`()` Returns the x-coordinate of the player's stick, in cm.
`int`	**`getStickY`**`()` Returns the y-coordinate of the player's stick, in cm.
`boolean`	**`hasPuck`**`()` Returns whether this player has the puck.
`boolean`	**`isLeftHanded`**`()` Returns whether this player is left handed.
`boolean`	**`isOpponent`**`()` Returns whether this player is your opponent in the game.

Methods inherited from interface hockey.api.IObject

`getHeading, getSpeed, getX, getY`

Method Detail

getStickAngle

```
public int getStickAngle()
```

Returns the stick relative angle relative to the player's heading, in degrees.

getStickR

```
public int getStickR()
```

Returns the stick distace from the player's centre, in cm.

---

getStickX

```
public int getStickX()
```

Returns the x-coordinate of the player's stick, in cm.

---

getStickY

```
public int getStickY()
```

Returns the y-coordinate of the player's stick, in cm.

---

getStick

```
public IObject getStick()
```

Returns the current position of the player's stick, as an `IObject`.

---

hasPuck

```
public boolean hasPuck()
```

Returns whether this player has the puck.

---

isOpponent

```
public boolean isOpponent()
```

Returns whether this player is your opponent in the game.

---

isLeftHanded

```
public boolean isLeftHanded()
```

Returns whether this player is left handed.

---

getIndex

```
public int getIndex()
```

Returns this player's index. The indices are:

- Your own
  - 0 - goal-keeper.
  - 1 - left defender.
  - 2 - right defender.
  - 3 - left forward.
  - 4 - right forward.
  - 5 - centre forward.
- Your opponent's
  - 6 - goal-keeper.
  - 7 - left defender.
  - 8 - right defender.
  - 9 - left forward.
  - 10 - right forward.
  - 11 - centre forward.

hockey.api

Interface IGoalKeeper

All Superinterfaces:
IObject, IPlayer

All Known Implementing Classes:
GoalKeeper

public interface **IGoalKeeper**
extends IPlayer

An `IObject` representing a goal-keeper on the ice.

See Also:
`IObject`, `IPlayer`

Method Summary

`int`	**`getGlide`**`()` Returns the goal-keeper's sidewards glide speed, in cm/s.
`IObject`	**`getGlove`**`()` Returns the current position of the goal-keeper's glove, as an `IObject`.

Methods inherited from interface hockey.api.IPlayer
`getIndex, getStick, getStickAngle, getStickR, getStickX, getStickY, hasPuck, isLeftHanded, isOpponent`

Methods inherited from interface hockey.api.IObject
`getHeading, getSpeed, getX, getY`

Method Detail

getGlide

```
public int getGlide()
```

Returns the goal-keeper's sidewards glide speed, in cm/s.

getGlove

```
public IObject getGlove()
```

Returns the current position of the goal-keeper's glove, as an `IObject`.

hockey.api

Class Position

```
java.lang.Object
  |
  +--hockey.api.Position
```

All Implemented Interfaces:
IObject

public class **Position**
extends java.lang.Object
implements IObject

An `IObject` implementation that represents a position on the ice. It may also have heading and speed.

See Also:
`IObject`

Constructor Summary

`Position(int x, int y)` Creates a new `Position` from a given position, with zero heading and speed.
`Position(int x, int y, int heading, int speed)` Creates a new `Position` from a given position, heading and speed.
`Position(IObject o)` Creates a new `Position` from a given `IObject`'s current position, heading and speed.

Method Summary

`int`	`getHeading()` Returns the absolute heading in degrees.
`int`	`getSpeed()` Returns the speed in cm/s.
`int`	`getX()` Returns the x-coordinate in cm.
`int`	`getY()` Returns the y-coordinate in cm.

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

Position

```
public Position(IObject o)
```

Creates a new `Position` from a given `IObject`'s current position, heading and speed.
Parameters:
`o` - the `IObject` that the position, heading and speed is copied from.

Position

```
public Position(int x,
                int y)
```

Creates a new `Position` from a given position, with zero heading and speed.

Parameters:

`x` - the x-coordinate of the position, in cm.
`y` - the y-coordinate of the position, in cm.

Position

```
public Position(int x,
                int y,
                int heading,
                int speed)
```

Creates a new `Position` from a given position, heading and speed.

Parameters:

`x` - the x-coordinate of the position, in cm.
`y` - the y-coordinate of the position, in cm.
`heading` - the `Position`'s heading, in degrees.
`speed` - the `Position`'s speed, in cm/s.

Method Detail

getX

```
public int getX()
```

Description copied from interface: `IObject`
Returns the x-coordinate in cm.

Specified by:
`getX` in interface `IObject`

Returns:
the x-coordinate, in cm.

---

getY

```
public int getY()
```

Description copied from interface: `IObject`
Returns the y-coordinate in cm.

Specified by:
`getY` in interface `IObject`

Returns:
the y-coordinate, in cm.

---

getHeading

```
public int getHeading()
```

Description copied from interface: `IObject`
Returns the absolute heading in degrees.

Specified by:
`getHeading` in interface `IObject`

Returns:
the absolute heading, in degrees.

---

getSpeed

```
public int getSpeed()
```

Description copied from interface: `IObject`
Returns the speed in cm/s.

Specified by:
`getSpeed` in interface `IObject`

Returns:
the speed, in cm/s.

hockey.api

Interface ITeam

public interface **ITeam**

Your team should implement this interface!

Provides information about a team, and fetches its players.

Method Summary

`GoalKeeper`	`getGoalKeeper()` Fetch the team's goal-keeper.
`int`	`getLuckyNumber()` Returns the team's LUCKY NUMBER in range 0-99999.
`Player`	`getPlayer(int index)` Fetch a player.
`java.awt.Color`	`getSecondaryTeamColor()` A team secondary color for the players' helmets.
`java.lang.String`	`getShortName()` A three to four letter long short name of the team.
`java.awt.Color`	`getTeamColor()` A team color for the players' shirts.
`java.lang.String`	`getTeamName()` A full team name.

Method Detail

getShortName

```
public java.lang.String getShortName()
```

A three to four letter long short name of the team.

getTeamName

```
public java.lang.String getTeamName()
```

A full team name.

getTeamColor

```
public java.awt.Color getTeamColor()
```

A team color for the players' shirts.

getSecondaryTeamColor

`public java.awt.Color getSecondaryTeamColor()`

A team secondary color for the players' helmets.

---

getGoalKeeper

`public GoalKeeper getGoalKeeper()`

Fetch the team's goal-keeper.

---

getLuckyNumber

`public int getLuckyNumber()`

Returns the team's LUCKY NUMBER in range 0-99999.

---

getPlayer

`public Player getPlayer(int index)`

Fetch a player.

Parameters:

`index` - the index of the player, in:

- 1 - Left defender.
- 2 - Right defender.
- 3 - Left forward.
- 4 - Right forward.
- 5 - Centre forward.

---

hockey.api

Class Player

```
java.lang.Object
  |
  +--hockey.api.Player
```

All Implemented Interfaces:
IObject, IPlayer, IPlayerControl

Direct Known Subclasses:
GoalKeeper

public abstract class **Player**
extends java.lang.Object
implements IPlayer, IPlayerControl

Every player in your team should extend this class!

It implements the `IPlayer` and `IPlayerControl` interfaces to give you full control and information of the player. Additionally it has some methods you must define to give information about your player, and some convenience methods for easier control.

There are seven abstract methods that must be implemented. Most important is the `step` method that controls your player.

- `getName`, `getNumber`, and `isLeftHanded` are used to give the game basic information about the player. The game will only call these once, upon startup.
- `init` is called after your player intelligence has been bound to an actual player. Here you can do initialisations which cannot be done in the constructor, such as `setAimOnStick`.
- Either `faceOff` or `penaltyShot` is called to notify you when the game is resumed after having been stopped for some reason. (Note that during penalty shots, only one of your players will be active!)
- `step` is by far the most important method of them all! This is where you control your player.

See Also:
`IObject`, `IPlayer`, `IPlayerControl`

Field Summary

Fields inherited from interface hockey.api.IPlayerControl
`ACCELERATION, MAX_SHOT_SPEED, MAX_SPEED, MAX_STICK_ANGLE, MAX_STICK_R, MAX_TURN_SPEED, MIN_STICK_ANGLE, MIN_STICK_R`

Constructor Summary

`Player()`

Method Summary

`void`	`faceOff()` Called before step when there is a face-off.
`boolean`	`getAimOnStick()` Returns whether the stick is used for aiming.
`IGoalKeeper`	`getGoalKeeper(int index)` Gets a goal-keeper from the game.
`int`	`getHeading()` Returns the absolute heading in degrees.
`int`	`getIndex()` Returns this player's index.
`abstract java.lang.String`	`getName()` The name of the player.
`abstract int`	`getNumber()`

	The number of the player.
IPlayer	getPlayer(int index) Gets a player from the game.
IPuck	getPuck() Gets the puck from the game.
int	getScore(boolean myScore) Gets the current score of one of the teams.
int	getSpeed() Returns the speed in cm/s.
IObject	getStick() Returns the current position of the player's stick, as an `IObject`.
int	getStickAngle() Returns the stick relative angle relative to the player's heading, in degrees.
int	getStickR() Returns the stick distace from the player's centre, in cm.
int	getStickX() Returns the x-coordinate of the player's stick, in cm.
int	getStickY() Returns the y-coordinate of the player's stick, in cm.
int	getTargetHeading() Returns the target heading, in degrees.
int	getTargetSpeed() Returns the target speed, in cm/s.
int	getTargetStickAngle() Returns the target stick relative angle from player's heading, in degrees.
int	getTargetStickR() Returns the target stick distance from player's centre, in degrees.
int	getTurnSpeed() Returns the turn speed, in degrees/s.
int	getX() Returns the x-coordinate in cm.
int	getY() Returns the y-coordinate in cm.
boolean	hasPuck() Returns whether this player has the puck.
void	init() Initialise the player.
abstract boolean	isLeftHanded() Whether the player is left-handed.
boolean	isOpponent() Returns whether this player is your opponent in the game.
void	moveStick(int dir, int dist) Move the stick to a given angle and distance relative to the player.
void	penaltyShot() Called before step when the player is about to take a penalty-shot.
void	setAimOnStick(boolean aos) Sets whether the stick or the body position should be used for controls.
void	setDebugPoint(int x, int y, java.awt.Color c) Sets a point displayed in the HDE.
void	setMessage(java.lang.String message) Sets the message displayed in the HDE.
void	setPlayerControl(IPlayer impl, IPlayerControl ctrl) This method is used internally to receive the `IPlayer` and `IPlayerControl` associated with this player.
void	shoot(int heading, int speed) Shoot the puck in a given absolute heading and speed.
void	shoot(int x, int y, int speed) Shoot the puck towards a given position with a given speed.
void	shoot(IObject o, int speed) Shoot the puck towards a given `IObject` with a given speed.
void	showDebugPoint(boolean show) Sets whether the debug point should be shown.
void	skate(int speed)

	Accelerate to a given speed.
void	**skate**(int x, int y, int speed) Accelerate to a given speed and turn towards a given position with maximum turn speed.
void	**skate**(IObject o, int speed) Accelerate to a given speed and turn towards a given `IObject` with maximum turn speed.
abstract void	**step**() Called every time step.
void	**turn**(int dir, int turnSpeed) Turn towards a given angle with a given turn speed.
void	**turn**(int x, int y, int turnSpeed) Turn towards a given position with a given turn speed.
void	**turn**(IObject o, int turnSpeed) Turn towards a given `IObject` with a given turn speed.

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

Player

```
public Player()
```

Method Detail

setPlayerControl

```
public final void setPlayerControl(IPlayer impl,
                                   IPlayerControl ctrl)
```

This method is used internally to receive the `IPlayer` and `IPlayerControl` associated with this player.

Parameters:
`impl` - the player's `IPlayer`.
`ctrl` - the player's `IPlayerControl`.

---

getNumber

```
public abstract int getNumber()
```

The number of the player. Numbers should be in the range 0-99.

---

getName

```
public abstract java.lang.String getName()
```

The name of the player.

---

isLeftHanded

```
public abstract boolean isLeftHanded()
```

Whether the player is left-handed.

Specified by:
`isLeftHanded` in interface `IPlayer`

---

init

```
public void init()
```

Initialise the player. (Here it is OK to use the API functions.)

---

faceOff

```
public void faceOff()
```

Called before step when there is a face-off.

---

penaltyShot

```
public void penaltyShot()
```

Called before step when the player is about to take a penalty-shot.

---

step

```
public abstract void step()
                   throws java.lang.Exception
```

Called every time step. This is where you put your player control code.

```
java.lang.Exception
```

---

getX

```
public int getX()
```

Description copied from interface: `IObject`
Returns the x-coordinate in cm.

Specified by:
`getX` in interface `IObject`

Returns:
the x-coordinate, in cm.

---

getY

```
public int getY()
```

Description copied from interface: `IObject`
Returns the y-coordinate in cm.

Specified by:
`getY` in interface `IObject`

Returns:
the y-coordinate, in cm.

---

getHeading

`public int getHeading()`

Description copied from interface: `IObject`
Returns the absolute heading in degrees.

Specified by:
`getHeading` in interface `IObject`

Returns:
the absolute heading, in degrees.

---

getSpeed

`public int getSpeed()`

Description copied from interface: `IObject`
Returns the speed in cm/s.

Specified by:
`getSpeed` in interface `IObject`

Returns:
the speed, in cm/s.

---

getStickAngle

`public int getStickAngle()`

Description copied from interface: `IPlayer`
Returns the stick relative angle relative to the player's heading, in degrees.

Specified by:
`getStickAngle` in interface `IPlayer`

---

getStickR

`public int getStickR()`

Description copied from interface: `IPlayer`
Returns the stick distace from the player's centre, in cm.

Specified by:
`getStickR` in interface `IPlayer`

---

getStickX

`public int getStickX()`

Description copied from interface: `IPlayer`
Returns the x-coordinate of the player's stick, in cm.

Specified by:
`getStickX` in interface `IPlayer`

---

getStickY

`public int getStickY()`

Description copied from interface: `IPlayer`
Returns the y-coordinate of the player's stick, in cm.

Specified by:
`getStickY` in interface `IPlayer`

---

getStick

`public IObject getStick()`

Description copied from interface: `IPlayer`
Returns the current position of the player's stick, as an `IObject`.

Specified by:
`getStick` in interface `IPlayer`

---

isOpponent

`public boolean isOpponent()`

Description copied from interface: `IPlayer`
Returns whether this player is your opponent in the game.

Specified by:
`isOpponent` in interface `IPlayer`

---

hasPuck

`public boolean hasPuck()`

Description copied from interface: `IPlayer`
Returns whether this player has the puck.

Specified by:
`hasPuck` in interface `IPlayer`

---

getIndex

`public int getIndex()`

Description copied from interface: `IPlayer`
Returns this player's index. The indices are:

- Your own
  - 0 - goal-keeper.
  - 1 - left defender.
  - 2 - right defender.
  - 3 - left forward.
  - 4 - right forward.
  - 5 - centre forward.
- Your opponent's
  - 6 - goal-keeper.
  - 7 - left defender.
  - 8 - right defender.
  - 9 - left forward.
  - 10 - right forward.
  - 11 - centre forward.

Specified by:
`getIndex` in interface `IPlayer`

---

getAimOnStick

```
public boolean getAimOnStick()
```

Description copied from interface: `IPlayerControl`
Returns whether the stick is used for aiming.

Specified by:
`getAimOnStick` in interface `IPlayerControl`

---

getTargetSpeed

```
public int getTargetSpeed()
```

Description copied from interface: `IPlayerControl`
Returns the target speed, in cm/s.

Specified by:
`getTargetSpeed` in interface `IPlayerControl`

---

getTargetHeading

```
public int getTargetHeading()
```

Description copied from interface: `IPlayerControl`
Returns the target heading, in degrees.

Specified by:
`getTargetHeading` in interface `IPlayerControl`

---

getTurnSpeed

```
public int getTurnSpeed()
```

Description copied from interface: `IPlayerControl`
Returns the turn speed, in degrees/s.

Specified by:
`getTurnSpeed` in interface `IPlayerControl`

---

getTargetStickAngle

```
public int getTargetStickAngle()
```

Description copied from interface: `IPlayerControl`
Returns the target stick relative angle from player's heading, in degrees.

Specified by:
`getTargetStickAngle` in interface `IPlayerControl`

---

getTargetStickR

```
public int getTargetStickR()
```

Description copied from interface: `IPlayerControl`
Returns the target stick distance from player's centre, in degrees.

Specified by:
`getTargetStickR` in interface `IPlayerControl`

---

setAimOnStick

```
public void setAimOnStick(boolean aos)
```

Description copied from interface: `IPlayerControl`
Sets whether the stick or the body position should be used for controls.

Specified by:
`setAimOnStick` in interface `IPlayerControl`

---

skate

```
public void skate(int speed)
```

Description copied from interface: `IPlayerControl`
Accelerate to a given speed.

Specified by:
`skate` in interface `IPlayerControl`

---

skate

```
public void skate(int x,
                  int y,
                  int speed)
```

Accelerate to a given speed and turn towards a given position with maximum turn speed.

---

skate

```
public void skate(IObject o,
                  int speed)
```

Accelerate to a given speed and turn towards a given `IObject` with maximum turn speed.

---

turn

```
public void turn(int dir,
                 int turnSpeed)
```

Description copied from interface: `IPlayerControl`
Turn towards a given angle with a given turn speed.

Specified by:
`turn` in interface `IPlayerControl`

---

turn

```
public void turn(int x,
                 int y,
                 int turnSpeed)
```

Turn towards a given position with a given turn speed.

---

turn

```
public void turn(IObject o,
                 int turnSpeed)
```

Turn towards a given `IObject` with a given turn speed.

---

moveStick

```
public void moveStick(int dir,
                      int dist)
```

Description copied from interface: `IPlayerControl`
Move the stick to a given angle and distance relative to the player.

Specified by:
`moveStick` in interface `IPlayerControl`

---

shoot

```
public void shoot(int heading,
                  int speed)
```

Description copied from interface: `IPlayerControl`
Shoot the puck in a given absolute heading and speed.

Specified by:
`shoot` in interface `IPlayerControl`

---

shoot

```
public void shoot(int x,
                  int y,
                  int speed)
```

Shoot the puck towards a given position with a given speed.

---

shoot

```
public void shoot(IObject o,
                  int speed)
```

Shoot the puck towards a given `IObject` with a given speed.

---

getScore

```
public int getScore(boolean myScore)
```

Description copied from interface: `IPlayerControl`
Gets the current score of one of the teams.

Specified by:
`getScore` in interface `IPlayerControl`

Parameters:
`myScore` - whether you want your own score or your opponent's score.

---

getPuck

```
public IPuck getPuck()
```

Description copied from interface: `IPlayerControl`
Gets the puck from the game.

Specified by:
`getPuck` in interface `IPlayerControl`

---

getGoalKeeper

```
public IGoalKeeper getGoalKeeper(int index)
```

Description copied from interface: `IPlayerControl`
Gets a goal-keeper from the game.

Specified by:
`getGoalKeeper` in interface `IPlayerControl`

Parameters:
`index` - the index of the goal-keeper:
- 0 - Your own goal-keeper.
- 6 - Your opponent's goal-keeper.

---

getPlayer

```
public IPlayer getPlayer(int index)
```

Description copied from interface: `IPlayerControl`
Gets a player from the game.

Specified by:
`getPlayer` in interface `IPlayerControl`

Parameters:
`index` - the index of the player. The indices are:
- Your own
  - 0 - goal-keeper.
  - 1 - left defender.
  - 2 - right defender.
  - 3 - left forward.
  - 4 - right forward.
  - 5 - centre forward.
- Your opponent's
  - 6 - goal-keeper.
  - 7 - left defender.
  - 8 - right defender.
  - 9 - left forward.
  - 10 - right forward.
  - 11 - centre forward.

---

setMessage

```
public void setMessage(java.lang.String message)
```

Description copied from interface: `IPlayerControl`
Sets the message displayed in the HDE. You should use this function instead of writing to `System.out`.

Specified by:
`setMessage` in interface `IPlayerControl`

Parameters:
`message` - the message.

---

setDebugPoint

```
public void setDebugPoint(int x,
                          int y,
                          java.awt.Color c)
```

Description copied from interface: `IPlayerControl`
Sets a point displayed in the HDE. This is very useful for debugging.

Specified by:
`setDebugPoint` in interface `IPlayerControl`

---

showDebugPoint

```
public void showDebugPoint(boolean show)
```

Description copied from interface: `IPlayerControl`
Sets whether the debug point should be shown.

Specified by:
`showDebugPoint` in interface `IPlayerControl`

---

hockey.api

Class GoalKeeper

```
java.lang.Object
  |
  +--hockey.api.Player
        |
        +--hockey.api.GoalKeeper
```

All Implemented Interfaces:
IGoalKeeper, IGoalKeeperControl, IObject, IPlayer, IPlayerControl

public abstract class **GoalKeeper**
extends Player
implements IGoalKeeper, IGoalKeeperControl

The goal-keeper in your team should extend this class!

Implements the `IGoalKeeper` and `IGoalKeeperControl` interfaces to give you full control and information of the goal-keeper. Additionally it has some methods you must define to give information about your goal-keeper, and some convenience methods for easier control.

Most important is the `step` method that controls your goal-keeper.

See Also:
`Player`, `IObject`, `IGoalKeeper`, `IPlayer`, `IGoalKeeperControl`, `IPlayerControl`

Field Summary

Fields inherited from interface hockey.api.IGoalKeeperControl
`MAX_GLIDE, MAX_THROW_SPEED`

Fields inherited from interface hockey.api.IPlayerControl
`ACCELERATION, MAX_SHOT_SPEED, MAX_SPEED, MAX_STICK_ANGLE, MAX_STICK_R, MAX_TURN_SPEED, MIN_STICK_ANGLE, MIN_STICK_R`

Constructor Summary

`GoalKeeper()`

Method Summary

`int`	`getGlide()` Returns the goal-keeper's sidewards glide speed, in cm/s.
`IObject`	`getGlove()` Returns the current position of the goal-keeper's glove, as an `IObject`.
`int`	`getTargetGlide()` Returns the target glide speed.
`void`	`glide(int glideSpeed)` Accelerate gliding sidewards to the given speed.
`void`	`setGoalKeeperControl(IGoalKeeper impl, IGoalKeeperControl ctrl)` This method is used internally to receive the `IGoalKeeper` and `IGoalKeeperControl` associated with this goal-keeper.
`void`	`skate(int x, int y, int speed)` Accelerate to a given speed and move towards a given position by skating or gliding, but without turning.

Methods inherited from class hockey.api.Player
`faceOff, getAimOnStick, getGoalKeeper, getHeading, getIndex, getName, getNumber, getPlayer, getPuck, getScore, getSpeed, getStick, getStickAngle, getStickR, getStickX, getStickY, getTargetHeading, getTargetSpeed, getTargetStickAngle, getTargetStickR, getTurnSpeed, getX, getY, hasPuck, init, isLeftHanded, isOpponent, moveStick, penaltyShot, setAimOnStick, setDebugPoint, setMessage, setPlayerControl, shoot, shoot, shoot, showDebugPoint, skate, skate, step, turn, turn, turn`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods inherited from interface hockey.api.IPlayer
`getIndex, getStick, getStickAngle, getStickR, getStickX, getStickY, hasPuck, isLeftHanded, isOpponent`

Methods inherited from interface hockey.api.IObject
`getHeading, getSpeed, getX, getY`

Methods inherited from interface hockey.api.IPlayerControl
`getAimOnStick, getGoalKeeper, getPlayer, getPuck, getScore, getTargetHeading, getTargetSpeed, getTargetStickAngle, getTargetStickR, getTurnSpeed, moveStick, setAimOnStick, setDebugPoint, setMessage, shoot, showDebugPoint, skate, turn`

Constructor Detail

GoalKeeper

```
public GoalKeeper()
```

Method Detail

setGoalKeeperControl

```
public final void setGoalKeeperControl(IGoalKeeper impl,
                                       IGoalKeeperControl ctrl)
```

This method is used internally to receive the `IGoalKeeper` and `IGoalKeeperControl` associated with this goal-keeper.

Parameters:
`impl` - the goal-keeper's `IGoalKeeper`.

---

getGlide

```
public int getGlide()
```

Description copied from interface: `IGoalKeeper`
Returns the goal-keeper's sidewards glide speed, in cm/s.

Specified by:
`getGlide` in interface `IGoalKeeper`

---

getGlove

```
public IObject getGlove()
```

Description copied from interface: `IGoalKeeper`
Returns the current position of the goal-keeper's glove, as an `IObject`.

Specified by:
`getGlove` in interface `IGoalKeeper`

getTargetGlide

```
public int getTargetGlide()
```

Description copied from interface: `IGoalKeeperControl`
Returns the target glide speed.

Specified by:
`getTargetGlide` in interface `IGoalKeeperControl`

glide

```
public void glide(int glideSpeed)
```

Description copied from interface: `IGoalKeeperControl`
Accelerate gliding sidewards to the given speed.

Specified by:
`glide` in interface `IGoalKeeperControl`

skate

```
public void skate(int x,
                  int y,
                  int speed)
```

Accelerate to a given speed and move towards a given position by skating or gliding, but without turning. Thus a goal-keeper's skate methods have different behaviour than a player's!

Overrides:
`skate` in class `Player`