LuckByDice

Jan 02, 2021

Project Information

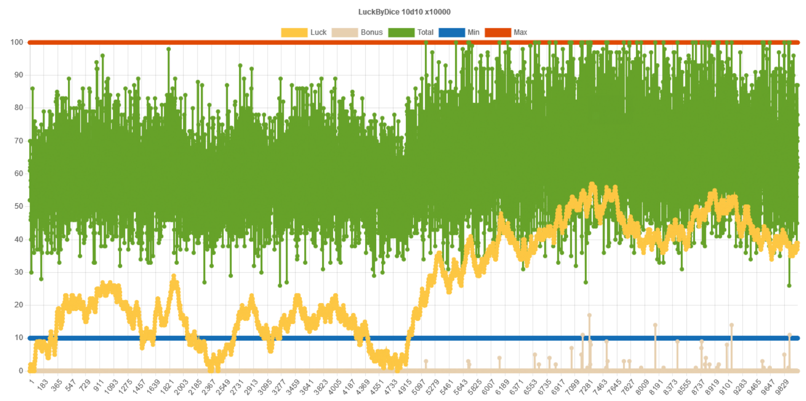
1	Installation	3
2	Contribute	5
3	Navigation	7
4	Indices and tables	25
	Index	27

Dice rolls add an element of chance and risk to game engines. They can be used to determine whether events within a game engine occur while still allowing skill progression and a skilled player to figure out their odds of performing the necessary actions.

This library simulates turns taken while rolling dice. But adds an ebb and flow to the outcome using a luck.

```php
require '../vendor/autoload.php';

use Ouxsoft\LuckByDice\Factory\TurnFactory;

$turn = TurnFactory::getInstance('d4,2d6,3d8+2,4d10*2,5d20+10*2,6d20-2,d%');
echo $turn->roll();

// based on the outcome luck increased (+1) or decreased (-1)
echo $turn->getLuck();

// our luck effects our next roll
echo $turn->roll();
```

This graph shows 10,000 consecutive *10d10* LuckByDice rolls. Notice how outcome impacts luck and vice versa.



CHAPTER 1

Installation

Get an instance of LuckByDice up and running in less than 5 minutes.

LuckByDice is available on Packagist.

Install with Composer:

```
composer require ouxsoft/luckbydice
```

That's it. You're done! Please take the rest of the time to read our docs.

CHAPTER 2

Contribute

- Issue Tracker: https://github.com/ouxsoft/LuckByDice/issues
- Source Code: https://github.com/ouxsoft/LuckByDice

CHAPTER 3

Navigation

3.1 Dice Notation Examples

Example of flipping a coin using dice notation:

```
d2
```

Example of rolling two four sided dice

```
2d4
```

Example of rolling a three six sided dice and adding two

```
3d6+2
```

Example of rolling four eight sided dice multiplying total by three

```
4d8*3
```

Example of rolling two ten sided dice and adding two to the total and multiplying by ten

```
2d10+2*8
```

Example of a percent or 100 side dice

```
d%
```

Example of rolling all above collections of dice at once

```
d2,2d4,3d6+2,4d8*3,2d10+2*8,d%
```

3.2 Glossary

3.2.1 Turn

A Turn rolls a Cup containing a Collection of Dice

3.2.2 Dice Notation

A dice notation is a string used to represent a Cup of dice.

3.2.3 Cup

A Cup holds and allows iteration of one ore more Collection of dice.

3.2.4 Collection

A Collection resides inside a Cup and contains one or more Dice with same amount of sides. A Collection also features a modifier and a multiplier for the roll outcome.

3.2.5 Dice

A dice has two or more sides. A dice can be roll. A dice features a value after rolling.

3.2.6 Luck

Luck simulates both the natual elusivity and ebb and flow of the concept of luck itself.

Luck is modified by and modifies the Dice's value.

```
Roll Outcome (x) = Round ((Random (0 - MaxRoll) + 1) * (Random (0 - Luck) *.01 + 1))
```

It is more probable that dice roll outcomes will increase when luck increases.

```
Luck Increases when Roll Outcome Percentage >= (1 / &phi;);
```

It is more probable that dice roll outcomes will decrease when luck decreases.

```
Luck Decreases when Roll Outcome Percentage <= (1 - (1 / &phi;))
```

A Dice's outcome may not exceed the sides of the dice no matter how lucky the player. A 6 sided dice will never yield 7. At most it may yield 6. At least it may yield 1.

3.2.7 Bonus

If applicable luck is higher than dice can absorb that value is represented as a bonus. A bonus is just a number which displays how much of a luck modifier could not be absorbed into dice because they were already at max value. Bonuses can be used for game engine to categorize the roll as a "Critical Hit", "Critical Hit x2", etc. if desired.

3.2.8 Algorithms

Algorithms are used to calculate luck.

3.3 API documentation

3.3.1 Contract namespace

AbstractFactoryInterface

Qualified name `Ouxsoft\LuckByDice\Contract\AbstractFactoryInterface`

`interface AbstractFactoryInterface`

`public makeCup``()`

`public makeLuck``()`

`public makeNotation``(`*Container $container*`)`

Parameters

- **`$container`** (`Container`) –

CollectionInterface

Qualified name `Ouxsoft\LuckByDice\Contract\CollectionInterface`

`interface CollectionInterface`

`public __construct``(`*int $amount, int $sides*[, *int $modifier, int $multiplier*]`)`

Parameters

- **`$amount`** (*`int`*) –
- **`$sides`** (*`int`*) –
- **`$modifier`** (*`int`*) – Default: `1`
- **`$multiplier`** (*`int`*) – Default: `1`

`public count``()`

`public getDice``()`

`public getMaxOutcome``()`

`public getMaxPotential``()`

`public getMinOutcome``()`

`public getMinPotential``()`

`public getModifier``()`

`public getMultiplier``()`

`public getNotation``()`

`public getOutcomePercent``()`

public getSides()

public getTotal()

public getValue()

public roll()

DiceInterface

Qualified name `Ouxsoft\LuckByDice\Contract\DiceInterface`

interface DiceInterface

public __construct(*$sides*)

Parameters

- **$sides** –

public getSides()

public roll()

LuckAdjustmentInterface

Qualified name `Ouxsoft\LuckByDice\Contract\LuckAdjustmentInterface`

interface LuckAdjustmentInterface

public run(*int $luck*[, *float $rollPercent*])

Parameters

- **$luck** (*int*) –
- **$rollPercent** (*float*) – Default: `0.5`

public setMax(*int $max*)

Parameters

- **$max** (*int*) –

public setMin(*int $min*)

Parameters

- **$min** (*int*) –

LuckInterface

Qualified name `Ouxsoft\LuckByDice\Contract\LuckInterface`

interface LuckInterface

public __construct([])

Parameters

- **$luck** (*int*) – Default: `0`

public get ()

public set (*int $luck*)

Parameters

- **$luck** (*int*) –

public update ([ ])

Parameters

- **$rollPercent** (*float*) – Default: `0.5`

NotationInterface

Qualified name `Ouxsoft\LuckByDice\Contract\NotationInterface`

interface NotationInterface

public __construct (*Cup $cup*)

Parameters

- **$cup** (Cup) –

public __toString ()

public get ()

public getSeparator ()

public set (*string $notation*)

Parameters

- **$notation** (*string*) –

public setSeparator (*$separator*)

Parameters

- **$separator** –

TurnInterface

Qualified name `Ouxsoft\LuckByDice\Contract\TurnInterface`

interface TurnInterface

public __construct (*Notation $notation*, *Cup $cup*, *Luck $luck*[, *string $expression*])

Parameters

- **$notation** (Notation) –
- **$cup** (Cup) –
- **$luck** (Luck) –
- **$expression** (*string*) – Default: `null`

public getCup ()

public getExtraBonus()

public getLuck()

public getMaxPotential()

public getMinPotential()

public getNotation()

public getTotal()

public roll()

public setLuck(*int $luck*)

Parameters

- **$luck** (*int*) –

3.3.2 Draw namespace

3.3.3 Factory namespace

ConcreteFactory

Qualified name `Ouxsoft\LuckByDice\Factory\ConcreteFactory`

Implements *AbstractFactoryInterface*

class ConcreteFactory

public makeCup()

public makeLuck()

public makeNotation(*Container $container*)

Parameters

- **$container** (Container) –

Container

Qualified name `Ouxsoft\LuckByDice\Factory\Container`

class Container

ContainerFactory

Qualified name `Ouxsoft\LuckByDice\Factory\ContainerFactory`

class ContainerFactory

static buildContainer(*AbstractFactoryInterface $abstractFactory*) → Container

Parameters

- **$abstractFactory** (AbstractFactoryInterface) –

Returns *`Container`* –

TurnFactory

Qualified name `Ouxsoft\LuckByDice\Factory\TurnFactory`

class TurnFactory

static getInstance([ ]) → Turn

Parameters

- **$notation** (*string*) – Default: `null`

Returns *`Turn`* –

3.3.4 LuckAdjustment namespace

AbstractLuckAdjustment

Qualified name `Ouxsoft\LuckByDice\LuckAdjustment\AbstractLuckAdjustment`

Implements *`LuckAdjustmentInterface`*

class AbstractLuckAdjustment

public getAdjustment([ ]) → int

Update luck based on percentage of roll outcome

Parameters

- **$rollPercent** (*float*) – Default: `0.5`

Returns int –

public getMax() → int

Get max

Returns int –

public getMin() → int

Get min

Returns int –

public getName() → string

Get name of adjustment class

Returns string –

public run(*int $currentLuck*[, *float $rollPercent*]) → int

Parameters

- **$currentLuck** (*int*) –
- **$rollPercent** (*float*) – Default: `0.5`

Returns int –

public setMax(*int $max*)

Set max

Parameters

- **`$max`** (*`int`*) –

Returns void

`public setMin`(*int $min*)
Set min

Parameters

- **`$min`** (*`int`*) –

Returns void

DefaultLuckAdjustment

Qualified name `Ouxsoft\LuckByDice\LuckAdjustment\DefaultLuckAdjustment`

Extends *`AbstractLuckAdjustment`*

`class DefaultLuckAdjustment`

`public getAdjustment`([ ]) → int
Update luck based on percentage of roll outcome

Parameters

- **`$rollPercent`** (*`float`*) – Default: `0.5`

Returns int –

`public getMax`() → int
Get max

Returns int –

`public getMin`() → int
Get min

Returns int –

`public getName`() → string
Get name of adjustment class

Returns string –

`public getPhi`() → float
Get Phi / The Golden Ratio

Returns float –

`public run`(*int $currentLuck*[, *float $rollPercent*]) → int

Parameters

- **`$currentLuck`** (*`int`*) –
- **`$rollPercent`** (*`float`*) – Default: `0.5`

Returns int –

`public setMax`(*int $max*)
Set max

Parameters

- **$max** (*int*) –

Returns void

public setMin (*int $min*)
Set min

Parameters

- **$min** (*int*) –

Returns void

FickleLuckAdjustment

Qualified name `Ouxsoft\LuckByDice\LuckAdjustment\FickleLuckAdjustment`

Extends *AbstractLuckAdjustment*

class FickleLuckAdjustment

public getAdjustment ([]) → int
Update luck based on percentage of roll outcome

Parameters

- **$rollPercent** (*float*) – Default: `0.5`

Returns int –

public getMax () → int
Get max

Returns int –

public getMin () → int
Get min

Returns int –

public getName () → string
Get name of adjustment class

Returns string –

public run (*int $currentLuck*[, *float $rollPercent*]) → int

Parameters

- **$currentLuck** (*int*) –
- **$rollPercent** (*float*) – Default: `0.5`

Returns int –

public setMax (*int $max*)
Set max

Parameters

- **$max** (*int*) –

Returns void

public setMin (*int $min*)
Set min

Parameters

- **`$min`** (*`int`*) –

Returns void

3.3.5 Collection

Qualified name `Ouxsoft\LuckByDice\Collection`

Implements *`CollectionInterface`*

`class Collection`

`public __construct` (*int $amount*, *int $sides*[, *int $modifier*, *int $multiplier*])

`Collection` constructor.

Parameters

- **`$amount`** (*`int`*) –
- **`$sides`** (*`int`*) –
- **`$modifier`** (*`int`*) – Default: `1`
- **`$multiplier`** (*`int`*) – Default: `1`

`public count` `()` → int

Returns int –

`public getBonus` `()` → int

Get bonus of dice without modifier or multiplier

Returns int –

`public getDice` `()` → array

Gets an array containing *`Dice`*

Returns array –

`public getMaxOutcome` `()` → int

Get max potential of outcome

Returns int –

`public getMaxPotential` `()` → int

Get the maximum potential of a collections

Takes into account maximum outcome, modifier, and multiplier

returns int –

`public` `Collection::`**`getMinOutcome`**`()` → int

Get min potential of outcome

Returns int –

`public` `Collection::`**`getMinPotential`**`()` → int

Get the minimum potential of a collections

Takes into account minimal outcome, modifier, and multiplier

returns int –

`public` `Collection::`**`getModifier`**`()` → int

Returns int –

public `Collection::`**`getMultiplier`**`()` → int

Returns int –

public `Collection::`**`getNotation`**`()` → string

Get the notation for the collection

Returns string –

public `Collection::`**`getOutcomePercent`**`()` → float

Compute percent outcome of previous roll

Convert dice outcomes to percent outcomes. *`Dice`* outcomes start counting at one. This formula works by starting the counts at 0. Example using 1d4: 1/4 = 0/3; 2/4 = 1/3; 3/4 = 2/3; 4/4 = 3/3

returns float –

public `Collection::`**`getSides`**`()` → int

Returns int –

public `Collection::`**`getTotal`**`([])` → int

Gets total value of each dice within *`Collection`* with modifier and multiplier applied

Parameters

- **`$adjustments`** (*bool*) – Default: `true`

Returns int –

public `Collection::`**`getValue`**`([])` → int

Get value of rolled dice

Parameters

- **`$adjustments`** (*bool*) – Default: `true`

Returns int –

public `Collection::`**`roll`**`()` → int

Roll each dice and returns Total

Returns int –

public `Collection::`**`setBonus`**`(`*int $amount*`)` → int

Distributes a new bonus across all dice

Parameters

- **`$amount`** (*int*) –

Returns int – returns the remaining bonus amount left to distribute

3.3.6 Cup

Qualified name `Ouxsoft\LuckByDice\Cup`

class Cup

public current`()`

Returns CollectionInterface

public empty`()`

public key() → int

Returns int –

public next()

public offsetExists(*$offset*) → bool

Parameters

- **$offset** –

Returns bool –

public offsetGet(*$offset*) → mixed

Parameters

- **$offset** –

Returns mixed –

public offsetSet(*$offset*, *$value*)

Parameters

- **$offset** –
- **$value** –

public offsetUnset(*$offset*)

Parameters

- **$offset** –

public reverse()

public rewind()

public valid() → bool

Returns bool –

3.3.7 Dice

Qualified name `Ouxsoft\LuckByDice\Dice`

Implements *DiceInterface*

class Dice

public __construct(*$sides*)
Dice constructor.

Parameters

- **$sides** –

public getBonus() → int

Returns int –

public getSides() → int

Returns int –

public getTotal()
The dices value with bonus from luck applied

Returns int|null

public getValue() → int

Returns int –

public roll() → int

Returns int –

public setBonus(*int $bonus*)

Parameters

- **$bonus** (*int*) –

public setSides(*int $sides*)

Parameters

- **$sides** (*int*) –

public setValue(*int $value*)

Parameters

- **$value** (*int*) –

3.3.8 Luck

Qualified name `Ouxsoft\LuckByDice\Luck`

Implements *LuckInterface*

class Luck

public __construct([])
Luck constructor.

Parameters

- **$luck** (*int*) – Default: `0`

public disable()
Disable luck

public enable()
Enable luck

public get() → int
Gets luck

Returns int –

public getActiveStatus() → bool
Get whether enabled or disabled

Returns bool –

public getAdjustment() → string
Get name of selected adjustment algorithm

Returns string –

public getApplicablePercent() → float

Get applicable luck as random percentage based on current luck

Returns float –

public modify(*int $number*) → int

Modifies a number based on current luck

Parameters

- **$number** (*int*) –

Returns int –

public set(*int $luck*)

Sets luck

Parameters

- **$luck** (*int*) –

public setAdjustment([ ])

Set the luck adjustment algorithm

Parameters

- **$algorithm** (*int*) – Default: `self::DEFAULT_ADJUSTMENT`

public update([ ])

Update luck based on percentage of roll outcome

Parameters

- **$rollPercent** (*float*) – Default: `0.5`

3.3.9 Notation

Qualified name `Ouxsoft\LuckByDice\Notation`

Implements *NotationInterface*

class Notation

public __construct(*Cup $cup*)

Notation constructor.

Parameters

- **$cup** (Cup) –

public __toString() -> *Notation::get*()

Returns *Notation::get()* –

public get() → string

Get cup notation

Returns string – "1d4+3*2,1d5,d5,10d5"

public getSeparator() → string

Returns string –

public set(*string $notation*)

Set cup notation

Parameters

- **`$notation`** (*`string`*) –

`public setSeparator` (*$separator*)

Parameters

- **`$separator`** –

`private decode` (*string $expression*)

Parameters

- **`$expression`** (*`string`*) –

`private encode` () → string

Returns string –

3.3.10 Parser

Qualified name `Ouxsoft\LuckByDice\Parser`

`class Parser`

`public run` (*$expression*) → array

Parameters

- **`$expression`** –

Returns array –

3.3.11 Turn

Qualified name `Ouxsoft\LuckByDice\Turn`

Implements *`TurnInterface`*

`class Turn`

`public __construct` (*Notation $notation, Cup $cup, Luck $luck*[*, string $expression*])

`Turn` constructor.

Parameters

- **`$notation`** (`Notation`) –
- **`$cup`** (`Cup`) –
- **`$luck`** (`Luck`) –
- **`$expression`** (*`string`*) – Default: `null`

Returns Turn::setByString()

`public getCup` () → Cup

Gets a *`Cup`* containing all Collections of *`Dice`*

Returns *`Cup`* –

public getExtraBonus() → int

Get extra bonuses that could not be absorbed by dice. This could be used for determining critical, etc. in game engines, etc.

Returns int –

public getLuck() → int

Get `Luck`

Returns int –

public getMaxPotential() → int

Get maximum potential of all Collections in `Cup`

Returns int –

public getMinPotential() → int

Get minimum potential of all Collections in `Cup`

Returns int –

public getNotation() → string

Get the dice notation for the entire cup

Returns string –

public getTotal() → int

Gets the Cups total which contains the outcome of all Collections of `Dice`

Returns int –

public roll() → int

Roll each dice group, update luck, and return outcome with luck modifier applied

Returns int – total

public setLuck(*int $luck*)

Set `Luck`

Parameters

- **$luck** (*int*) –

3.4 Code of Conduct

3.4.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

3.4.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism

- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

3.4.3 Our Responsibilities

Maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

3.4.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

3.4.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the Code of Conduct Committee at <conduct@ouxsoft.com>. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The Code of Conduct Committee is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

3.4.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://www.contributor-covenant.org/version/1/4/code-of-conduct.html

CHAPTER 4

Indices and tables

- genindex
- search

Index

Symbols

__construct() (Collection method), 16
__construct() (CollectionInterface method), 9
__construct() (Dice method), 18
__construct() (DiceInterface method), 10
__construct() (Luck method), 19
__construct() (LuckInterface method), 10
__construct() (Notation method), 20
__construct() (NotationInterface method), 11
__construct() (Turn method), 21
__construct() (TurnInterface method), 11
__toString() (Notation method), 20
__toString() (NotationInterface method), 11

A

AbstractFactoryInterface (interface), 9
AbstractLuckAdjustment (class), 13

B

buildContainer() (ContainerFactory method), 12

C

Collection (class), 16
CollectionInterface (interface), 9
ConcreteFactory (class), 12
Container (class), 12
ContainerFactory (class), 12
count() (Collection method), 16
count() (CollectionInterface method), 9
Cup (class), 17
current() (Cup method), 17

D

decode() (Notation method), 21
DefaultLuckAdjustment (class), 14
Dice (class), 18
DiceInterface (interface), 10
disable() (Luck method), 19

E

empty() (Cup method), 17
enable() (Luck method), 19
encode() (Notation method), 21

F

FickleLuckAdjustment (class), 15

G

get() (Luck method), 19
get() (LuckInterface method), 10
get() (Notation method), 20
get() (NotationInterface method), 11
getActiveStatus() (Luck method), 19
getAdjustment() (AbstractLuckAdjustment method), 13
getAdjustment() (DefaultLuckAdjustment method), 14
getAdjustment() (FickleLuckAdjustment method), 15
getAdjustment() (Luck method), 19
getApplicablePercent() (Luck method), 20
getBonus() (Collection method), 16
getBonus() (Dice method), 18
getCup() (Turn method), 21
getCup() (TurnInterface method), 11
getDice() (Collection method), 16
getDice() (CollectionInterface method), 9
getExtraBonus() (Turn method), 21
getExtraBonus() (TurnInterface method), 11
getInstance() (TurnFactory method), 13
getLuck() (Turn method), 22
getLuck() (TurnInterface method), 12
getMax() (AbstractLuckAdjustment method), 13
getMax() (DefaultLuckAdjustment method), 14
getMax() (FickleLuckAdjustment method), 15
getMaxOutcome() (Collection method), 16
getMaxOutcome() (CollectionInterface method), 9
getMaxPotential() (Collection method), 16
getMaxPotential() (CollectionInterface method), 9
getMaxPotential() (Turn method), 22
getMaxPotential() (TurnInterface method), 12

getMin() (AbstractLuckAdjustment method), 13
getMin() (DefaultLuckAdjustment method), 14
getMin() (FickleLuckAdjustment method), 15
getMinOutcome() (Collection method), 16
getMinOutcome() (CollectionInterface method), 9
getMinPotential() (Collection method), 16
getMinPotential() (CollectionInterface method), 9
getMinPotential() (Turn method), 22
getMinPotential() (TurnInterface method), 12
getModifier() (Collection method), 16
getModifier() (CollectionInterface method), 9
getMultiplier() (Collection method), 17
getMultiplier() (CollectionInterface method), 9
getName() (AbstractLuckAdjustment method), 13
getName() (DefaultLuckAdjustment method), 14
getName() (FickleLuckAdjustment method), 15
getNotation() (Collection method), 17
getNotation() (CollectionInterface method), 9
getNotation() (Turn method), 22
getNotation() (TurnInterface method), 12
getOutcomePercent() (Collection method), 17
getOutcomePercent() (CollectionInterface method), 9
getPhi() (DefaultLuckAdjustment method), 14
getSeparator() (Notation method), 20
getSeparator() (NotationInterface method), 11
getSides() (Collection method), 17
getSides() (CollectionInterface method), 10
getSides() (Dice method), 18
getSides() (DiceInterface method), 10
getTotal() (Collection method), 17
getTotal() (CollectionInterface method), 10
getTotal() (Dice method), 18
getTotal() (Turn method), 22
getTotal() (TurnInterface method), 12
getValue() (Collection method), 17
getValue() (CollectionInterface method), 10
getValue() (Dice method), 19

K

key() (Cup method), 18

L

Luck (class), 19
LuckAdjustmentInterface (interface), 10
LuckInterface (interface), 10

M

makeCup() (AbstractFactoryInterface method), 9
makeCup() (ConcreteFactory method), 12
makeLuck() (AbstractFactoryInterface method), 9
makeLuck() (ConcreteFactory method), 12
makeNotation() (AbstractFactoryInterface method), 9
makeNotation() (ConcreteFactory method), 12
modify() (Luck method), 20

N

next() (Cup method), 18
Notation (class), 20
NotationInterface (interface), 11

O

offsetExists() (Cup method), 18
offsetGet() (Cup method), 18
offsetSet() (Cup method), 18
offsetUnset() (Cup method), 18

P

Parser (class), 21

R

reverse() (Cup method), 18
rewind() (Cup method), 18
roll() (Collection method), 17
roll() (CollectionInterface method), 10
roll() (Dice method), 19
roll() (DiceInterface method), 10
roll() (Turn method), 22
roll() (TurnInterface method), 12
run() (AbstractLuckAdjustment method), 13
run() (DefaultLuckAdjustment method), 14
run() (FickleLuckAdjustment method), 15
run() (LuckAdjustmentInterface method), 10
run() (Parser method), 21

S

set() (Luck method), 20
set() (LuckInterface method), 11
set() (Notation method), 20
set() (NotationInterface method), 11
setAdjustment() (Luck method), 20
setBonus() (Collection method), 17
setBonus() (Dice method), 19
setLuck() (Turn method), 22
setLuck() (TurnInterface method), 12
setMax() (AbstractLuckAdjustment method), 13
setMax() (DefaultLuckAdjustment method), 14
setMax() (FickleLuckAdjustment method), 15
setMax() (LuckAdjustmentInterface method), 10
setMin() (AbstractLuckAdjustment method), 14
setMin() (DefaultLuckAdjustment method), 15
setMin() (FickleLuckAdjustment method), 15
setMin() (LuckAdjustmentInterface method), 10
setSeparator() (Notation method), 21
setSeparator() (NotationInterface method), 11
setSides() (Dice method), 19
setValue() (Dice method), 19

T

Turn (class), 21

TurnFactory (class), 13
TurnInterface (interface), 11

U

update() (Luck method), 20
update() (LuckInterface method), 11

V

valid() (Cup method), 18