

# Progetto P2 - Model V 0.3.1

## Table of contents

- Meta-documentazione
  - Convezioni
    - Nomenclatura
    - Sintassi e highlight nei documenti
    - Convenzioni di sintassi del codice
  - Cambiamenti rispetto alla V0.2
  - Update/Rework/Addis rispetto alla V0.3
    - Metodologia di esposizione dei contenuti
    - Tabella di tracking
    - Note aggiuntive sui cambiamenti
- Gerarchia di classi: visione d'insieme
  - Classe `Item`
    - Campi dati
    - Definizione dei campi dati
    - Metodi e comportamento
  - Classe `Spell`
    - Campi dati
    - Definizione dei campi dati
    - Metodi e comportamento
  - Classe `Generic`
    - Campi dati
    - Definizione dei campi dati
    - Metodi e comportamento
  - Classe `Equip`
    - Campi dati
    - Definizione dei campi dati
    - Metodi e comportamento
  - Classe `MagicGeneric`
    - Campi dati
    - Definizione dei campi dati
    - Metodi e comportamento
  - Classe `Weapon`
    - Campi dati
    - Definizione dei campi dati
    - Metodi e comportamento
  - Classe `Armor`
    - Campi dati
    - Definizione dei campi dati
    - Metodi e comportamento

## Meta-documentazione

# Convezioni

## Nomenclatura

Possibili convenzioni di nomenclatura:

- I nomi assegnati alle classi sono in lingua inglese con lettera maiuscola. (es: `Item`)
- I nomi assegnati alle variabili sono in lingua inglese composti unicamente da lettere minuscole. Se composti devono essere intervallati dal carattere '\_' (es: `weapon_name`);
- I nomi dei metodi in lingua inglese. Se il metodo e' composto da più termini il primo ha lettera minuscola, tutte le iniziali dei successivi hanno lettera maiuscola (es. `setName`);
- Devono essere dichiarate le direttive d'uso che si vogliono usare per i metodi appartenenti a un namespace (es. se voglio non dover riscrivere `std::endl` ogni volta che lo utilizzo, devo notificare che viene dichiarata quella direttiva d'uso e in quale file);
- TUTTI i nomi che vengono utilizzati devono essere quanto più concisi possibile, prendendo come lunghezza indicativa tra i 5 e i 7 caratteri (non tassativo ma utile)

## Sintassi e highlight nei documenti

- I nomi delle classi vengono evidenziate in modo da rendere chiaro che si parla di classi (es. `Item` ).
- Per ogni classe sono definiti Campi dati e il significato degli stessi
- I campi dati delle classi sono presentati in forma tabulare
- Le descrizioni per i campi dati sono presentate in forma tabulare espansa

## Convenzioni di sintassi del codice

- I costruttori utilizzano le liste di inizializzazione che non sono riportate nel documento (TODO)

## Cambiamenti rispetto alla V0.2

Nella V02 della gerarchia c'erano alcuni problemi fondamentali di struttura e contenuto per i dati:

1. Le classi `Armor` e le sottoclassi di `Weapon` contenevano una grande quantità di informazioni simili, portando a ripetizione del codice in fase di definizione. Per ovviare al problema, vengono disposte come figlie di un'unica interfaccia che racchiude quelle caratteristiche comuni, delegando alle sottoclassi l'onere di implementare aspetti più specifici. Questo permette inoltre di unificare tutti quegli oggetti che sono utilizzati come armamentario/equipaggiamento sotto un'unica interfaccia, alleggerendo la gerarchia. Questo cambiamento comporta una serie di vantaggi:
  - viene introdotta una distinzione tra oggetti *riproducibili* e *unici*. Per esemplificare: una spada lunga e' un'arma generica, sia che essa possieda proprietà magiche sia che non ne possieda. Una spada che esiste come oggetto unico e dotato di proprietà uniche può essere introdotta attraverso l'implementazione di una sottoclasse che ne incapsuli i comportamenti unici.
  - nel caso si desideri introdurre una nuova tipologia di oggetti che ricadono nella categoria degli equipaggiamenti sarà sufficiente introdurre una nuova sottoclasse di `EquipItem` che ne incapsuli le caratteristiche.
2. `Generic` (sottoclasse di `Item` ), nella gerarchia precedente fungeva da interfaccia per eventuali sottotipi di oggetto (tipi specializzati). A conti fatti però tutto quello che deve fare e' permettere la definizione di

oggetti che non hanno proprieta' particolari al di fuori dell'essere esse stesse un oggetto. In questa iterazione non e' piu' interfaccia ma classe concreta che funge solo da implementazione elementare per la classe Item e che assolve al compito di permettere la definizione di oggetti "elementari", che non hanno alcuna proprieta' particolare (es. sasso, corda, torcia).

3. Come conseguenza del punto precedente, diventava difficoltoso definire in che modo si differenziassero gli oggetti che possedevano proprieta' magiche (e che quindi assumevano una rilevanza maggiore) da quelli che invece erano definiti senza quelle proprieta'. Per ovviare al problema viene introdotta la classe `MagicItem` che appunto si prende carico di definire le proprieta' generiche di un oggetto magico.

## Update/Rework/Adds rispetto alla V0.3

### Metodologia di esposizione dei contenuti

*Rework* indica un cambiamento sostanziale in alcuni aspetti del conenuto o del contenuto nella sua interezza.

*Update* indica cambiamenti minori che non impattano in maniera considerevole il lavoro svolto in versioni precedenti.

*Adds* indica l'aggiunta di aspetti/componenti non presenti in versioni precedenti.

### Tabella di tracking

TIPOLOGIA	CONTENUTO
ADDS	aggiunta firma metodi e loro comportamento
REWORK	modifica progettuale alla classe Spell che diventa concreta
REWORK	modifica progettuale alla classe MagicItem che diventa concreta
REWORK	modifica dei nomi della classi
UPDATE	aggiunta campo dati "num" alla classe Generic
UPDATE	aggiunta campo dati "is_magic" alla classe EquipItem
UPDATE	aggiunta campo dati "equipped" alla classe EquipItem
UPDATE	aggiunta campo dati "side_material" alla classe EquipItem
REWORK	modifica della classe Armor

### Note aggiuntive sui cambiamenti

REWORK di Spell	non è realmente necessario andare a ridefinire ogni volta i metodi che la classe Spell deve fornire, a conti fatti il comportamento dei metodi propri della classe rimane sempre lo stesso. E' nelle classi figlie che viene utilizzato per valutare parametri propri dell'oggetto.
REWORK di MagicItem	così come per Generic viene definita un'unica classe dalla quale possono essere derivate specializzazioni là dove lo si ritiene necessario, MagicItem diventa una classe i cui oggetti rappresentano oggetti magici di qualsiasi natura e che può essere estesa

nel caso si vogliano creare specializzazioni o sottoinsieme di questi.

---

<b>REWORK di nomi delle classi</b>	<ul style="list-style-type: none"><li>• EquipItem -&gt; Equip</li><li>• MagicItem -&gt; MagicGeneric</li></ul>
--	--

---

---

## Gerarchia di classi: visione d'insieme

La gerarchia si compone di:

- 1 classi base virtuale pura
  - Item
- 1 classe base concreta
  - Spell
- 3 classi di prima derivazione
  - Generic (concreta)
  - Equip (virtuale)
  - MagicGeneric (concreta)

Da EquipItem derivano due classi concrete

- Weapon
- Armor

che rivestono un ruolo meno importante nella gerarchia generale e sono utili ai fini del progetto.



### Sidenote

La classe **Inventario** identifica il container di item

La classe **Personaggio** identifica l'oggetto a cui viene associato il container

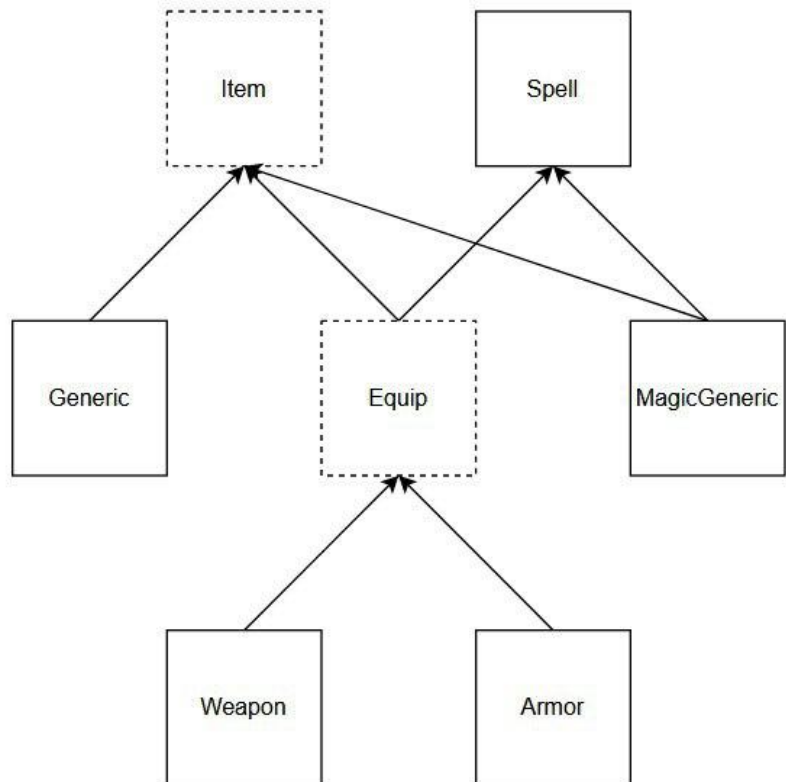
Non viene specificato nulla di aggiuntivo qui circa queste due classi

Le due classi base permettono la definizione di qualunque oggetto rappresentabile e utilizzabile in un gioco di ruolo moderno.

L'idea fondante consiste nel derivare da queste due classi qualora si volesse creare una nuova categoria di oggetti, oppure raffinare le categorie già presenti implementando classi che sono figlie di EquipItem, Generic o MagicItem.

Personaggio

Inventario



## Classe Item

Classe base virtuale pura che rappresenta le caratteristiche generiche di un oggetto.

### Campi dati

Campo Dati	Tipo
name	string
value	int
weight	int
material	string

### Definizione dei campi dati

**name** Il nome dell'oggetto.

**val** Il valore dell'oggetto

**weight** Il peso dell'oggetto

**material** Il materiale di cui é composto l'oggetto

## Metodi e comportamento

### Costruttori

---

<b>Item(string, int, int, string)</b>	Costruttore ridefinito che assegna su ogni campo dati
---	---

---

### Metodi propri non virtuali

---

<b>void setName(const string&amp;)</b>	Modifica il nome dell'oggetto.
--	--------------------------------

---

<b>void setVal(const int&amp;)</b>	Modifica il valore dell'oggetto.
--	----------------------------------

---

<b>int getWeight() const</b>	Recupera il peso dell'oggetto.
--------------------------------------	--------------------------------

---

### Metodi propri virtuali

---

<b>virtual string getMaterial() const</b>	Recupera il materiale di cui e' composto l'oggetto
---	--

---

### Metodi virtuali puri

---

<b>virtual string getName() const=0</b>	Recupera le informazione sul nome dell'oggetto. Virtuale perche' (ad esempio) negli equipaggiamenti risulta utile riferire il nome dell'oggetto e il materiale di cui e' composto in un unica sede (es. Spada in acciaio e non solo Spada)
---	--


---

<b>virtual int getVal() const=0</b>	Recupera il valore dell'oggetto. Virtuale per coprire il caso di oggetti di equipaggiamento o magici in cui potrebbe essere utile ridefinirlo in modo che il valore tenga in considerazione altri aspetti dell'oggetto stesso
---	---

---

## Classe Spell

Classe base concreta che rappresenta la magia come proprietà che può essere associata ad un item.

 **N.B.**

Da osservare che la non presenza di proprietà magiche in un Equip Item viene rappresentata come una magia nulla e che quindi viene richiesto un costruttore nullo

Campi dati

Campo Dati	Tipo
spell_name	string
spell_type	string
spell_level	int
spell_power	int
spell_duration	int

Definizione dei campi dati

<b>spell_name</b>	Il nome della magia
<b>spell_type</b>	Il tipo di magia (es. Fuoco)
<b>spell_level</b>	Il livello della magia
<b>spell_power</b>	L'intensità della magia
<b>spell_duration</b>	La durata della magia, espressa in turni

Metodi e comportamento

Costruttori

<b>Spell()</b>	Costruttore nullo, per rispettare la nota che si trova nell'overview della classe
<b>Spell(string, string, int, int, int): campi dati propri</b>	Costruttore che assegna su ogni campo dati
<b>Spell(string, string, int,</b>	Costruttore che assegna su ogni campo dati tranne duration

**int): campi  
dati propri**

---

## **Metodi propri non virtuali**

---

<b>string</b> <b>getSpellName()</b> <b>const</b>	Recupera il nome della magia
--	------------------------------

---

<b>string</b> <b>getSpellType()</b> <b>const</b>	Recupera il tipo (elemento) della magia
--	---

---

<b>int</b> <b>getSpellLevel()</b> <b>const</b>	Recupera il livello della magia
--	---------------------------------

---

<b>int</b> <b>getSpellPower()</b> <b>const</b>	Recupera l'indice di potere della magia
--	---

---

<b>int</b> <b>getSpellDuration()</b> <b>const</b>	Recupera il tempo di durata della magia
---	---

---

<b>void</b> <b>setSpellName(const</b> <b>string&amp;)</b>	Modifica il nome della magia
---	------------------------------

---

<b>void</b> <b>setSpellType(const</b> <b>string&amp;)</b>	Modifica il tipo (elemento) della magia
---	---

---

<b>void</b> <b>setSpellLevel(const</b> <b>int&amp;)</b>	Modifica il livello della magia
---	---------------------------------

---

<b>void</b> <b>setSpellPower(const</b> <b>int&amp;)</b>	Recupera l'indice di potere della magia
---	---

---

<b>void</b> <b>setSpellDuration(const</b> <b>int&amp;)</b>	Modifica il tempo di durata della magia
--	---

---

---

## **Classe Generic**



Classe derivata unicamente da Item che rappresenta un item che non possiede proprieta' magiche.

## Campi dati

Campo Dati	Tipo
num	int

## Definizione dei campi dati

---

<b>num</b>	Se si possiedono più oggetti identici a questo, num > 1
------------	---

---

## Metodi e comportamento

### Costruttori

---

<b>Generic(string int, int, string, num): campi dati propri</b>	Costruttore ridefinito che assegna su ogni campo dati
---	---

---

### Metodi di cui viene fatto Override

---

<b>override string getName() const</b>	Recupera il nome dell'oggetto.
--	--------------------------------

---

<b>override int getVal() const</b>	Recupera il valore dell'oggetto.
--	----------------------------------

---

<b>override int getWeight() const</b>	Recupera il peso dell'oggetto.
---	--------------------------------

---

<b>ostream&amp; operator&lt;&lt;(ostr const Generic&amp;)</b>	Operatore di output
---	---------------------

---

<b>istream&amp;  operator&gt;&gt; (istream&amp;,</b>	Operatore di input
--	--------------------

---

Generic&)

## Metodi propri non virtuali

<b>void</b> <b>setNum(const</b> <b>int&amp;)</b>	Modifica il campo dati num
<b>int</b> <b>getNum()</b> <b>const</b>	Recupera il valore del campo dati num

## Classe Equip

Classe derivata da Item e Magic che rappresenta il concetto generico di equipaggiamento da battaglia

### Campi dati

Campo Dati	Tipo
equip_power	int
type	string
rarity	string
side_material	string
is_magic	bool
equipped	bool

### Definizione dei campi dati

<b>equip_power</b>	Il valore di potenza dell'equipaggiamento
<b>type</b>	Il tipo di equipaggiamento, distinguibile in: <ul style="list-style-type: none"><li>• leggero</li><li>• pesante</li></ul>
<b>rarity</b>	La rarità dell'arma che ha come valori possibili: <ul style="list-style-type: none"><li>• common</li><li>• rare</li><li>• epic</li><li>• legendary</li></ul>

! **N.B.**

Questa classe indica una generalizzazione di un oggetto, quindi non ha senso andare a definire un livello di rarita' **Unique** per identificare un oggetto non categorizzabile (es. Excalibur e' un oggetto unico, non riproducibile). Nel caso in cui si volesse creare un oggetto unico si deve creare una nuova classe che deriva da `EquipItem` e che identifica quello specifico oggetto.

<b>side_material</b>	Eventuali materiali aggiuntivi di cui è composta l'arma
<b>is_magic</b>	Parametro che traccia la presenza o meno di proprietà magiche nell'oggetto
<b>equipped</b>	Parametro che traccia se l'oggetto e' equipaggiato o meno

## Metodi e comportamento

### Costruttori

<pre>Equip(int,       string,       string,       string,       bool, bool,       string, int,       int, string,       string,       string, int,       int, int):     Item(...),     Spell(...), elenco campi dati propri</pre>	Costruttore che assegna su ogni campo dati e sui sottogetti <i>Item</i> e <i>Spell</i>
---	--

<pre>Equip(int,       string,       string,       string,       bool, bool,       string, int,        int,       string):     Item(...),     Spell(), elenco campi dati propri</pre>	Costruttore che assegna su tutti i campo dati e SOLO sul sottogetto <i>Item</i>
--	---

## Metodi di cui viene fatto Override

<b>override</b> <b>string</b> <b>GetName()</b> <b>const</b>	Recupera il nome dell'oggetto e lo concatena al materiale e al fatto che sia un equipaggiamento Magico oppure no (es. Spada magica in diamante).
<b>override int</b> <b>GetVal()</b> <b>const</b>	Recupera il valore dell'oggetto e lo aumenta sulla base del materiale e sulla presenza o meno di proprieta' magiche.
<b>override</b> <b>string</b> <b>getMaterial()</b> <b>const</b>	Recupera le informazioni sui materiali (Item::material e EquipItem::side_material) di cui è composto l'oggetto.

## Metodi di cui NON viene fatto Override



### Motivazione

Teoricamente, non ho necessita' di creare oggetti del tipo EquipItem ma solo sue derivazioni, quindi non implemento input e output perche' non necessari

<b>override</b> <b>ostream&amp;</b> <b>operator&lt;&lt;(ostr</b> <b>const</b> <b>Generic&amp;)</b>	Operatore di output
<b>override</b> <b>istream&amp;</b> <b>operator&gt;&gt;</b> <b>(istream&amp;,</b> <b>Generic&amp;)</b>	Operatore di input

## Metodi propri non virtuali

<b>int</b> <b>getEquipPower()</b> <b>const</b>	Recupera il valore di potenza dell'oggetto
<b>string</b> <b>getType()</b>	Recupera il tipo dell'oggetto

<b>const</b>	
<b>string</b> <b>getRarity()</b> <b>const</b>	Recupera il livello di rarita' dell'oggetto
<b>bool</b> <b>getIsEquipped()</b> <b>const</b>	Recupera lo status di equipaggiamento dell'oggetto
<b>bool</b> <b>getIsMagic()</b> <b>const</b>	Recupera lo status magico dell'oggetto
<b>void</b> <b>setEquipPower(c</b> <b>in&amp;)</b>	Setta il valore di potenza dell'oggetto
<b>void</b> <b>setType(const</b> <b>string&amp;)</b>	Settaa il tipo dell'oggetto
<b>void</b> <b>setRarity(const</b> <b>string&amp;)</b>	Setta il livello di rarita' dell'oggetto
<b>virtual void</b> <b>setIsEquipped(c</b> <b>bool&amp;)</b>	Setta lo status di equipaggiamento dell'oggetto
<b>void</b> <b>setIsMagic(cons</b> <b>bool&amp;)</b>	Setta lo status magico dell'oggetto

## Classe MagicGeneric

La classe MagicItem identifica un oggetto generico che pero' gode di proprieta' magiche e che quindi richiede una specifica rappresentazione.

### Campi dati

Campo Dati	Tipo
range	int
description	string

### Definizione dei campi dati

<b>range</b>	L' area di effetto della magia
<b>description</b>	La descrzione dell' effetto della magia

## Metodi e comportamento

### Costruttori

<b>MagicItem()</b>	Costruttore vuoto
<b>MagicItem(string, int, int, string, string, string, int, int, int, int):</b> <b>Item(...),</b> <b>Spell(...)</b>	Costruttore che assegna su tutti i campi dati
<b>MagicItem(string, int, int, string, string, string, int, int, int, int):</b> <b>Item(...),</b> <b>Spell(...)</b>	Costruttore che assegna su tutti i campi dati tranne duration (usa il secondo costruttore di Spell)

### Metodi virtuali ridefiniti

<b>virtual string getName() const</b>	Recupera il nome dell'oggetto concatenandolo alla sua proprietà magica
<b>virtual int getVal() const</b>	Recupera il valore dell'oggetto.

### Metodi propri non virtuali

<b>int getRange() const</b>	Recupera il range a cui può agire l'oggetto/Incantesimo
-----------------------------	---

<b>string</b> <b>getDescription(</b> <b>const</b>	Recupera la descrizione dell'oggetto
<b>void</b> <b>setRange(const</b> <b>int&amp;)</b>	Modifica il range
<b>void</b> <b>setDescription(</b> <b>string&amp;)</b>	Modifica la descrizione dell'oggetto
<b>override</b> <b>ostream&amp;</b> <b>operator&lt;&lt;(ostr</b> <b>const</b> <b>Generic&amp;)</b>	Operatore di output
<b>override</b> <b>istream&amp;</b> <b>operator&gt;&gt;</b> <b>(istream&amp;,</b> <b>Generic&amp;)</b>	Operatore di input

## Classe Weapon

### Campi dati

Campo Dati	Tipo
melee_type_dmg	string
ranged_type_dmg	string
min_range	int
max_range	int

### Definizione dei campi dati

<b>melee_type_dmg</b>	Il tipo di danno che l'arma e' in grado di infliggere corpo a corpo
<b>ranged_type_dmg</b>	Il tipo di danno che l'arma e' in grado di infliggere dalla distanza, se vi e' modo di farne.
<b>min_range</b>	La distanza minima a cui e' possibile colpire
<b>max_range</b>	La distanza massima a cui e' possibile colpire

---

## Metodi e comportamento

### Costruttori

---

<pre>Weapon(int,         string,         string,         string,         bool, bool,         string, int,         int, string,         string,         string, int,         int, int,         string,         string, int,         int):     Equip(...),     campi dati     propri</pre>	Costruttore che assegna su tutti i campi dati
--	---

---

<pre>Weapon(int,         string,         string,         string,         bool, bool,         string, int,         int, string,         string,         string, int,         int):     Equip(...),     campi dati     propri</pre>	Costruttore che assegna sui campi dati dai Equip
---	--

---

### Metodi propri non virtuali

---

<pre>override ostream&amp; operator&lt;&lt;(ostr            const Generic&amp;)</pre>	Operatore di output
---	---------------------

---

<pre>override</pre>	Operatore di input
---------------------	--------------------



```

    istream&
operator>>
(istream&,
Generic&)

```

<b>string getMeleeTypeDmg const</b>	Recupera il tipo di danno che l'arma e' in grado di infliggere corpo a corpo
<b>string getRangedTypeDn const</b>	Recupera tipo di danno che l'arma e' in grado di infliggere dalla distanza.
<b>int getMinRange() const</b>	Recupera la distanza minima a cui è possibile colpire
<b>int getMaxRange() const</b>	Recupera la distanza massima a cui è possibile colpire
<b>string getMeleeTypeDmg const</b>	Recupera il tipo di danno che l'arma e' in grado di infliggere corpo a corpo
<b>string getRangedTypeDn const</b>	Recupera tipo di danno che l'arma e' in grado di infliggere dalla distanza.
<b>int getMinRange() const</b>	Recupera la distanza minima a cui è possibile colpire
<b>int getMaxRange() const</b>	Recupera la distanza massima a cui è possibile colpire
<b>void setMeleeTypeDmg</b>	Setta il tipo di danno che l'arma e' in grado di infliggere corpo a corpo
<b>void setRangedTypeDn</b>	Recupera tipo di danno che l'arma e' in grado di infliggere dalla distanza.
<b>void setMinRange()</b>	Recupera la distanza minima a cui è possibile colpire
<b>void setMaxRange()</b>	Recupera la distanza massima a cui è possibile colpire

**Metodi che grazie a dio non sono getter o setter**

---

<b>bool</b> <b>isMagic()</b>	Recupera le informazioni circa la natura magica (o meno) dell'arma
---------------------------------	--

---

<b>int</b> <b>weaponDmg()</b>	Recupera il danno che l'arma è in grado di infliggere, tenendo conto del fatto che sia magica o meno
----------------------------------	--

---

## Classe Armor

Classe derivata da EquipItem che rappresenta un armatura nella sua interezza.

### Campi dati

Campo Dati	Tipo
damaged	bool

### Definizione dei campi dati

---

<b>damaged</b>	Traccia lo stato di integrità o meno dell'armatura
----------------	--

---

## Metodi e comportamento

### Costruttori

---

<b>Weapon(int, string, string, string, bool, bool, string, int, int, string, string, string, int, int, int, bool):</b> <b>Equip(...),</b> <b>campi dati</b> <b>propri</b>	Costruttore che assegna su tutti i campi dati
--	---

---

### Metodi propri non virtuali

---

<b>bool</b> <b>getDamaged()</b>	Recupera le informazioni circa lo stato di integrità dell'armatura
------------------------------------	--

**const**

---

**void**      Setta le informazioni circa lo stato di integrità dell'armatura  
**setDamaged( )**

---