## Cose importantissime da tenere a mente:

- Quando si scrive del codice, la sintassi è FONDAMENTALE. Significa che tutte le parole, i punti e virgola, le virgole, i punti e le parentesi devono essere scritte con la massima attenzione!
    - ogni istruzione finisce con un punto e virgola ;
    - ogni parentesi aperta deve essere, prima o poi, chiusa { }
    - a proposito di parentesi, le parentesi graffe - cioè { } - sono diverse dalle tonde!
    - le maiuscole e le minuscole sono importanti!
    - gli spazi e l'andare a capo, invece, non sono importanti. Potete andare a capo quante volte volete, potete andare a capo anche in mezzo a una riga, e il risultato non cambierà. Potete mettere quanti spazi volete, ma non potete spezzare una parola con gli spazi.
    - Le righe che iniziano con // sono commenti e non vengono letti dal pc, sono solo per gli esseri umani.
- I dubbi possono venire a qualsiasi programmatore, non possiamo sapere tutto! Sai come si dice in questi casi? GIYF, Google Is Your Friend. Cioè, prova a cercare su Google il tuo dubbio! Saper cercare su Google in modo intelligente è una cosa importantissima.
- Testa spesso i cambiamenti che hai fatto, ti aiuterà moltissimo nel caso in cui tu abbia scritto qualcosa di sbagliato a capire dove è l'errore.

# Le porte logiche

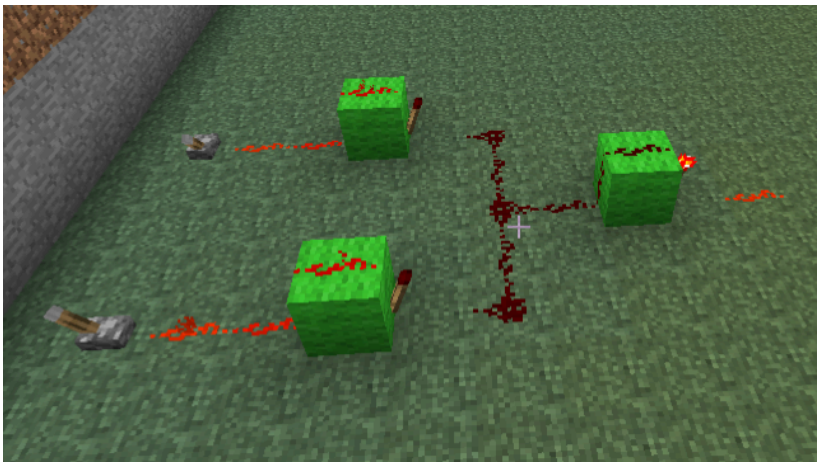
Alla base dell'importanza di Minecraft c'è la possibilità di rappresentare le porte logiche all'interno del gioco. Una porta logica è un circuito che calcola un'operazione matematica. Sembra una cosa complicata, ma tutti voi l'avete fatto almeno una volta se avete giocato con la redstone!

Le porte logiche sono il mattoncino base per costruire TUTTO. Ce ne sono alcune basilari, che fanno operazioni semplicissime. Queste porte basilari, se collegate insieme in modi intelligenti, costituiscono circuiti più complicati. Basta avere la possibilità di ricreare questi semplici pezzettini per poter costruire tutto il resto: avete mai visto calcolatrici, computer che funzionano all'interno di minecraft, programmi che girano? La possibilità di costruire le porte logiche rende Minecraft un linguaggio di programmazione Turing-completo, ovvero, detto in parole semplici, un linguaggio di programmazione che può simulare il funzionamento di qualsiasi macchina.

Le porte logiche lavorano con la logica booleana. Significa che lavorano con solo due valori. Possiamo dare la definizione che vogliamo a questi due valori, queste sono le definizioni più comuni (sono equivalenti):

| | | | |
|---|---|---|---|
| 1 | vero | acceso |  |
| 0 | falso | spento |  |

Un esempio di porta logica in Minecraft:

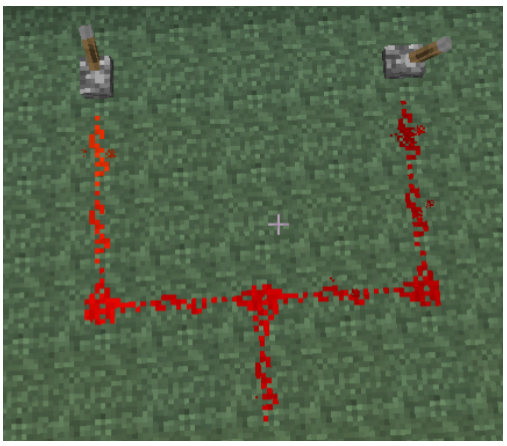


Le parti di redstone accese corrispondono a 1, vero, acceso.
Questa porta fa in modo che, se sono accesi i due input (a sinistra) - solo se sono accesi entrambi - l'output (a destra) sarà anch'esso acceso.

Proviamo a scandire bene cosa fa la porta:

*Se è acceso il primo interruttore **e** il secondo interruttore*

Se è vera la prima parte della frase e anche la seconda, allora tutta la frase è vera, 1, accesa. Come l'output, la parte più a destra del circuito. Questa, infatti, è una porta AND, che in inglese significa "e"!



Guardiamo anche questa porta: quando è vero che l'uscita è accesa?
*Se è acceso il primo interruttore **oppure** il secondo interruttore*

Infatti questa è una porta OR, che significa, ovviamente, oppure.

## Le tabelle di verità

Le tabelle di verità sono un modo per capire rapidamente cosa fa una porta.
Si scrivono su una tabella tutti i modi in cui possono essere gli input e il risultato che ne deriva.
Prendiamo l'esempio della prima porta che abbiamo visto: AND.
Abbiamo due interruttori, quindi facciamo due colonne per gli interruttori:

| *Interruttore 1* | *Interruttore 2* |
|---|---|
| Acceso | Acceso |
| Acceso | Spento |
| Spento | Acceso |
| Spento | Spento |

Nelle colonne degli interruttori scriviamo tutte le possibili combinazioni di come possono stare gli interruttori (uno acceso e uno spento? entrambi spenti? e così via…). Poi ci chiediamo: che succede se sono entrambi accesi? E scriviamo il risultato nella colonna apposita, così:

| *Interruttore 1* | *Interruttore 2* | *risultato (Interruttore 1 AND Interruttore 2)* |
|---|---|---|
| Acceso | Acceso | Acceso |
| Acceso | Spento | Spento |
| Spento | Acceso | Spento |
| Spento | Spento | Spento |

Troverete scritte tutte le tabelle di verità delle porte logiche di base, ma questo tipo di tabella ci aiuta moltissimo quando dobbiamo combinare più porte logiche insieme.
Per semplificare le cose userò "1" e "0" al posto di acceso e spento, e "A" e "B" al posto di scrivere Interruttore 1 e Interruttore 2.

## Tutte le porte di base

Per rappresentare come si costruiscono in minecraft userò i seguenti simboli:


## NOT

La porta logica più semplice è la porta NOT. Si fa con un solo interruttore.

Esempio a parole:

*L'interruttore **non** è abbassato*

Schema:

Tabella di verità:

| *A* | *NOT A* |
|---|---|
| 1 | 0 |
| 0 | 1 |

## AND

Esempio a parole:

*Il primo **e** il secondo interruttore sono abbassati*

Schema:

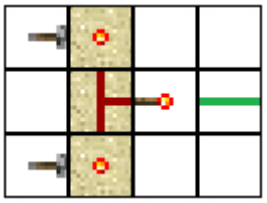


Tabella di verità:

| *A* | *B* | *A AND B* |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

## OR

Esempio a parole:

*Il primo interruttore è abbassato **oppure** il secondo interruttore è abbassato*

Schema:

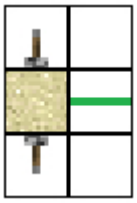


Tabella di verità:

| *A* | *B* | *A OR B* |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

## XOR

XOR è il nome per “Exclusive OR”. Come potete notare nell’OR, se i due interruttori sono abbassati, allora il risultato è acceso. Lo XOR è diverso dall’or solo per il fatto che se entrambi sono abbassati, il risultato sarà spento.

Schema:

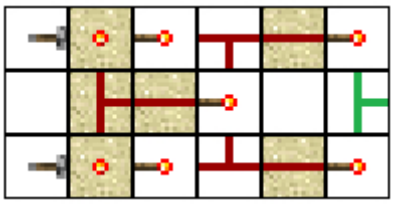


Tabella di verità:

| *A* | *B* | *A XOR B* |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

## NAND

NAND è il contrario di AND.

Schema:

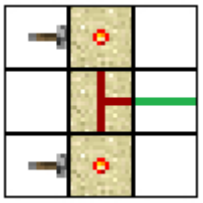


Tabella di verità:

| *A* | *B* | *A NAND B* |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 1 |

## NOR

NOR è il contrario di OR.

Schema:

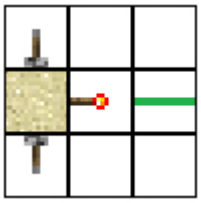


Tabella di verità:

| *A* | *B* | *A NOR B* |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

# Comandi

Struttura di un comando:

*/azione @<selettore>[<argomento>=<valore>,<argomento>=<valore>,…]*

Selettori:

| | |
|---|---|
| @p | giocatore più vicino |
| @r | giocatore random |
| @a | tutti i giocatori |
| @e | tutte le entità |

Argomenti:

| | |
|---|---|
| x, y, z | coordinate |
| r, rm | raggio attorno a te |
| m | modalità di gioco |
| c | quantità |
| l, lm | experience level (max, min) |
| score_*name* | max score |
| score_*name*_min | min score |
| team | team |
| name | name |
| entity | name |
| dx, dy, dz | volume dimensions |
| rx, rxm | vertical rotation (max, min) |
| ry, rym | horizontal rotation (max, min) |
| type | entity type |

esempio: kill @e[type=Creeper,r=20] uccide tutti i creeper nel raggio di 20 blocchi.

Alcune azioni utili:

/clear <player> - cancella tutto l'inventario
/give <player> <item_id> <amount> - da un oggetto a un giocatore. L'item_id deve essere una parola del tipo "minecraft:planks"
/kill <player> - ucccide un mostro/giocatore
/setblock <x> <y> <z> <TileName> - trasforma il blocco alle coordinate x,y,z nel blocco TileName (ad es. "minecraft:stone")
/summon <entityName> - spawna un entity di nome entityName (ad es. "Bat", "WitherSkull", "PrimedTNT")
/time set <value> - setta il tempo a value (ad es. "day", "night", "7000")
/tp <player> <x> <y> <z> - teletrasporta il giocatore alle coordinate x,y,z
/xp <value> - dona un numero uguale a value di punti esperienza al giocatore

"tilde", ovvero questo simbolo: ~ serve a far partire le coordinate dal punto in cui si trova il giocatore che invia il comando.

Esempio: fill
fill *<x1> <y1> <z1> <x2> <y2> <z2> <TileName>* [*oldBlockHandling*]
fill riempie lo spazio indicato con i blocchi del tipo specificato.
TileName deve essere sostituito dall'id di un blocco, come "minecraft:stone"

oldBlockHandling può specificare il modo di sostituire i vecchi blocchi: se ci scriviamo “destroy”, sostituisce tutti i vecchi blocchi presenti all’interno dello spazio indicato, se ci scriviamo “keep” li mantiene, se ci scriviamo “hollow”, creerà un box vuoto all’interno.
Alcuni esempi di utilizzo di fill:
/fill ~-3 ~-3 ~-3 ~3 ~-1 ~3 minecraft:water 0 sostituisce con blocchi d’acqua i blocchi subito sotto al giocatore
/fill ~-3 ~ ~-4 ~3 ~4 ~4 minecraft:planks 2 hollow costruisce un box delle dimensioni di una casa attorno al giocatore

Alcuni block id utili:

| Dec | Hex | Name | Block |
|---|---|---|---|
| 0 | 0 | minecraft:air | Air |
| 1 | 1 | minecraft:stone | Stone S B |
| 2 | 2 | minecraft:grass | Grass Block |
| 3 | 3 | minecraft:dirt | Dirt S B |
| 4 | 4 | minecraft:cobblestone | Cobblestone |
| 5 | 5 | minecraft:planks | Wood Planks S B |
| 6 | 6 | minecraft:sapling | Sapling S B |
| 7 | 7 | minecraft:bedrock | Bedrock |
| 8 | 8 | minecraft:flowing_water | Water S |
| 9 | 9 | minecraft:water | Stationary Water S |
| 10 | A | minecraft:flowing_lava | Lava S |
| 11 | B | minecraft:lava | Stationary Lava S |
| 12 | C | minecraft:sand | Sand S B |
| 13 | D | minecraft:gravel | Gravel |
| 14 | E | minecraft:gold_ore | Gold Ore |
| 15 | F | minecraft:iron_ore | Iron Ore |

| Dec | Hex | Name | Block |
|---|---|---|---|
| 16 | 10 | minecraft:coal_ore | Coal Ore |
| 17 | 11 | minecraft:log | Wood S B |
| 18 | 12 | minecraft:leaves | Leaves S B |
| 19 | 13 | minecraft:sponge | Sponge S B |
| 20 | 14 | minecraft:glass | Glass |
| 21 | 15 | minecraft:lapis_ore | Lapis Lazuli Ore |
| 22 | 16 | minecraft:lapis_block | Lapis Lazuli Block |
| 23 | 17 | minecraft:dispenser | Dispenser S E |
| 24 | 18 | minecraft:sandstone | Sandstone S B |
| 25 | 19 | minecraft:noteblock | Note Block E |
| 26 | 1A | minecraft:bed | Bed I S |
| 27 | 1B | minecraft:golden_rail | Powered Rail S |
| 28 | 1C | minecraft:detector_rail | Detector Rail S |
| 29 | 1D | minecraft:sticky_piston | Sticky Piston S |
| 30 | 1E | minecraft:web | Cobweb |
| 31 | 1F | minecraft:tallgrass | Grass S B |

| Dec | Hex | Name | Block |
|---|---|---|---|
| 32 | 20 | minecraft:deadbush | Dead Bush |
| 33 | 21 | minecraft:piston | Piston S |
| 34 | 22 | minecraft:piston_head | Piston Head S |
| 35 | 23 | minecraft:wool | Wool S B |
| 36 | 24 | minecraft:piston_extension | Block moved by Piston E |
| 37 | 25 | minecraft:yellow_flower | Dandelion |
| 38 | 26 | minecraft:red_flower | Poppy S B |
| 39 | 27 | minecraft:brown_mushroom | Brown Mushroom |
| 40 | 28 | minecraft:red_mushroom | Red Mushroom |
| 41 | 29 | minecraft:gold_block | Block of Gold |
| 42 | 2A | minecraft:iron_block | Block of Iron |
| 43 | 2B | minecraft:double_stone_slab | Double Stone Slab S B |
| 44 | 2C | minecraft:stone_slab | Stone Slab S B |
| 45 | 2D | minecraft:brick_block | Bricks |
| 46 | 2E | minecraft:tnt | TNT |
| 47 | 2F | minecraft:bookshelf | Bookshelf |

Altri block id si possono trovare qui: http://minecraft.gamepedia.com/Data_values#Block_IDs

Il comando summon:

summon <*EntityName*> [*x*] [*y*] [*z*]

Fa spawnare un’entità che corrisponde al nome indicato nella posizione indicata.

Alcuni nomi di entità utili:

| Id | Hex | Egg | Entity | Savegame ID |
|---|---|---|---|---|
| | | | Hostile mobs | |
| 50 | 32 | | Creeper | Creeper |
| 51 | 33 | | Skeleton<br>Wither Skeleton | Skeleton |
| 52 | 34 | | Spider | Spider |
| 53 | 35 | - | Giant | Giant |
| 54 | 36 | | Zombie<br>Zombie Villager | Zombie |
| 55 | 37 | | Slime | Slime |
| 56 | 38 | | Ghast | Ghast |
| 57 | 39 | | Zombie Pigman | PigZombie |
| 58 | 3A | | Enderman | Enderman |
| 59 | 3B | | Cave Spider | CaveSpider |
| 60 | 3C | | Silverfish | Silverfish |
| 61 | 3D | | Blaze | Blaze |
| 62 | 3E | | Magma Cube | LavaSlime |
| 63 | 3F | - | Ender Dragon | EnderDragon |
| 64 | 40 | - | Wither | WitherBoss |
| 66 | 42 | | Witch | Witch |
| 67 | 43 | | Endermite | Endermite |
| 68 | 44 | | Guardian<br>Elder Guardian | Guardian |
| 69 | 45 | | Shulker | Shulker |
| 101 | 65 | | Killer Rabbit | Rabbit |

| | | | | | |
|---|---|---|---|---|---|
| 65 | 41 | | | Bat | Bat |
| 90 | 5A | | | Pig | Pig |
| 91 | 5B | | | Sheep | Sheep |
| 92 | 5C | | | Cow | Cow |
| 93 | 5D | | | Chicken | Chicken |
| 94 | 5E | | | Squid | Squid |
| 95 | 5F | | | Wolf | Wolf |
| 96 | 60 | | | Mooshroom | MushroomCow |
| 97 | 61 | | - | Snow Golem | SnowMan |
| 98 | 62 | | | Ocelot | Ozelot |
| 99 | 63 | | - | Iron Golem | VillagerGolem |
| 100 | 64 | | | Horse | EntityHorse |
| 101 | 65 | | | Rabbit | Rabbit |
| | | | | NPCs | |
| 120 | 78 | | | Villager | Villager |

## ExampleMod.java:

Questa è la classe base da cui si parte per costruire una mod.
Avete la classe già pronta dentro le cartelle src/main/java/com/example/examplemod.
Niente paura se la vostra classe è leggermente differente! Potete aggiungere e togliere tutti i pezzi che sono diversi.

```
package com.example.examplemod;
import net.minecraft.block.Block;
// ... (lista dei vari import)

@Mod(modid = ExampleMod.MODID, version = ExampleMod.VERSION)

public class ExampleMod{
// ExampleMod è il nome della classe e deve essere sempre uguale al nome
// del file!

      public static final String MODID = "examplemod";
      public static final String VERSION = "1.0";

      @EventHandler
      public void preinit(FMLPreInitializationEvent event){
       Qui inseriremo il codice per fare la nostra mod
      }

      @EventHandler
      public void init(FMLInitializationEvent event){
      Qui inseriremo il codice per fare la nostra mod
      }

}
```

## Aggiungere una ricetta:

Cos'è un ItemStack? Un ItemStack è una pila di oggetti: 
Dentro Minecraft viene considerato come un oggetto solo.

Le ricette si aggiungono dentro ExampleMod.java, cerca questo punto del codice:

```
@EventHandler
	public void init(FMLInitializationEvent event){
	Qui aggiungo la mia ricetta!
	}
```

## Ricette senza forma:

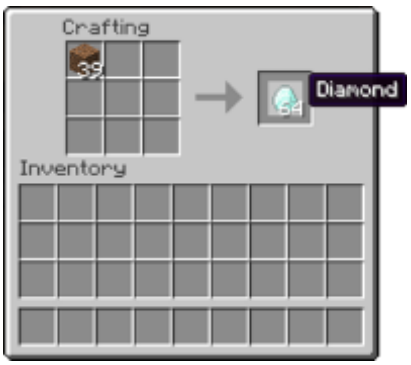
Le ricette senza forma sono quelle ricette che non richiedono che gli ingredienti siano disposti in posizioni particolari.
Si fa così:

```
GameRegistry.addShapelessRecipe(RISULTATO, INGREDIENTI);
```

GameRegistry.addShapelessRecipe è la funzione del gioco a cui chiediamo di aggiungere una ricetta. Tra le parentesi subito dopo la funzione bisogna mettere due ItemStack, il primo descrive l'oggetto che otteniamo dalla ricetta, e il secondo l'oggetto di cui si ha bisogno per crearlo.
Esempio:

```
GameRegistry.addShapelessRecipe(    new ItemStack(Item.diamond,64),    new ItemStack(Block.Dirt));

                                    un ItemStack di 64 diamanti è      Un singolo dirt è necessario
                                    quello che riceveremo come         per ottenere 64 diamanti
                                    risultato del crafting
```




## Ricette con la forma:

Magari vogliamo creare una ricetta che richiede una particolare disposizione degli ingredienti per essere realizzata. Ecco come si può fare:

```
GameRegistry.addRecipe(new ItemStack(Block.cobblestone),
    "AA",
    "AA",
    'A', new ItemStack(Block.dirt));
```

Questo pezzo di codice dice che quattro dirt, messi a forma di quadrato, ci faranno ottenere un cobblestone.

Come? Nella prima riga c'è scritto il risultato della ricetta. Nella seconda e nella terza c'è una 'A' che per noi significa che al posto della A ci dovrà essere un dirt. L'ultima riga specifica che 'A' rappresenta un blocco di dirt. Attenzione: al posto della A si può mettere qualsiasi lettera! Basta usare sempre la stessa per blocchi dello stesso tipo.

Cosa succederebbe se volessimo fare una ricetta più complessa?

```
GameRegistry.addRecipe(
    new ItemStack(Block.stone),
    "xyx",
    "y y",
    "xyx",
    'x', new ItemStack(Block.dirt),
    'y', new ItemStack(Block.gravel));
```

In questa ricetta abbiamo usato due lettere: 'x' indica un blocco di dirt, mentre 'y' indica un blocco di gravel. Lo spazio, invece, indica che quella posizione va lasciata vuota. Ecco come viene la ricetta:


## Ricette di smelting:

Le ricette di smelting si aggiungono in modo molto simile alle altre, come questa:

```
GameRegistry.addSmelting(Blocks.obsidian, new ItemStack(Items.diamond, 2), 1F);
```

L'unica cosa che hanno in più è l'ultimo valore, un numero (seguito da una F!) che indica quanta esperienza darà la creazione dell'oggetto.

## Come deve essere il codice di ExampleMod.java dopo aver aggiunto una ricetta:

```
package com.example.examplemod;
import net.minecraft.block.Block;
// … (lista dei vari import)

@Mod(modid = ExampleMod.MODID, version = ExampleMod.VERSION)

public class ExampleMod{

      public static final String MODID = "examplemod";
      public static final String VERSION = "1.0";

      @EventHandler
      public void preinit(FMLPreInitializationEvent event){
       Qui inseriremo il codice per fare la nostra mod
      }

      @EventHandler
      public void init(FMLInitializationEvent event){
      GameRegistry.addRecipe(
       new ItemStack(Block.stone),
       "xyx",
       "y y",
       "xyx",
       'x', new ItemStack(Block.dirt),
       'y', new ItemStack(Block.gravel));
      }
}
```

# Le classi e i blocchi
## Cos'è una classe?
Ripercorriamo quello che ci siamo detti sulle classi, vi ho chiesto:

Che cos'è un veicolo?

Le risposte che mi avete dato sono state:

una cosa che si muove
una cosa che consuma carburante
una cosa che si può comandare
una cosa che trasporta le persone

Allora, prendiamo per esempio la macchina e l'aereo, che sono due veicoli.
Cos'è che rende l'aereo un veicolo speciale? E la macchina?

vola
è grandissimo

ha il volante
ha quattro ruote

Dunque, possiamo dire che la macchina e l'aereo sono entrambi veicoli, hanno tutte le caratteristiche dei veicoli, ma hanno alcune caratteristiche che li rendono veicoli speciali, che hanno sia le caratteristiche dei veicoli, sia alcune caratteristiche proprie.

Nella programmazione a oggetti, un veicolo è una **classe.**
La macchina e l'aereo sono sue **sottoclassi**, che ereditano le stesse caratteristiche dei veicoli, e in più ne hanno alcune loro.
Una macchina in particolare, invece, come la macchina di mia mamma o la macchina di tuo papà, si chiama **istanza** di un veicolo, perché quando ne parlo mi riferisco a una macchina in particolare, e non a tutte le macchine.

Facciamo la stessa cosa con i blocchi di Minecraft: che cos'è un blocco?

una cosa che occupa uno spazio
una cosa che si può rompere
una cosa che si vede

Adesso descriviamo un blocco di dirt e un blocco di diamond ore:

si rompe facilmente
se ci cammini si sente il prato calpestato
ci cresce l'erba sopra

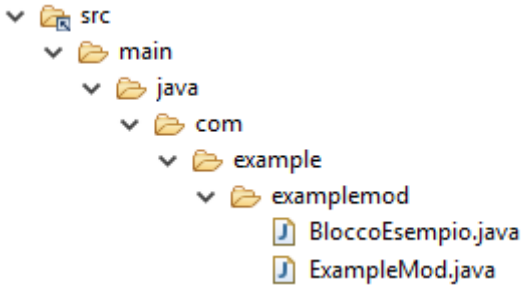
si rompe difficilmente
se lo rompi escono dei diamanti

Il blocco di dirt e il blocco di diamond ore sono dei blocchi, ma hanno delle caratteristiche in più: sono sottoclassi della classe blocco!

## Aggiungere un blocco:

Vogliamo creare il nostro primo blocco personalizzato.
Visto che la descrizione di un blocco standard e del suo funzionamento è lunghissima e complicatissima e qualcuno l'ha già scritta al posto nostro, vogliamo prendere lo stesso codice ed estenderlo, senza bisogno di riscriverlo. Per fare ciò creeremo una sottoclasse della classe blocco.

Prima di tutto bisogna creare una nuova classe: sulla sinistra, in Eclipse, troviamo il Navigator, che ci permette di esplorare tutte le cartelle del progetto su cui stiamo lavorando. Per ora ci interessa solo la cartella in cui si trova già ExampleMod.java:




Per creare un nuovo file che conterrà il nostro nuovo blocco, clicchiamo con il tasto destro sulla cartella "examplemod", selezioniamo "new" e poi "Class". Ci si aprirà una finestra nella quale ci sarà un campo "Name" da riempire con il nome del blocco che desideriamo fare.
Come si vede nell'immagine sopra, il mio blocco si chiama "BloccoEsempio.java".
Questo deve essere come è dentro (per ora):

```
package com.example.examplemod;

public class BloccoEsempio{

}
```

<u>Cose essenziali da aggiungere a questo file:</u>

- subito dopo "public class BloccoEsempio", prima della parentesi graffa, bisogna scrivere "extends Block". Questo ci permette di dire che il nostro BloccoEsempio appartiene alla classe dei blocchi, e di conseguenza ne eredita tutte le caratteristiche.

  ```
  public class BloccoEsempio extends Block{
  ```

- dentro le parentesi di BloccoEsempio bisogna aggiungere il costruttore, ovvero una funzione che spiega cosa succede quando il nostro nuovo blocco viene aggiunto al gioco. Si scrive così:

  ```
  public class BloccoEsempio extends Block{

      public BloccoEsempio(){
          super(Material.rock);
          GameRegistry.registerBlock(this, "ciao");
          setCreativeTab(CreativeTabs.tabBlock);
          setUnlocalizedName("ciao");
      }

  }
  ```

Le linee dentro il costruttore sono essenziali per aggiungere il primo blocco!
Ecco cosa significano:

```
super(Material.rock);
```

Sta semplicemente dicendo alla classe Blocco che il materiale di cui è costituito il nostro blocco è la roccia, e dunque BloccoEsempio erediterà tutte le caratteristiche della roccia. La roccia può essere sostituita con altri tipi di materiali.

```
GameRegistry.registerBlock(this, "ciao");
```

Dice al gioco di aggiungere all'elenco dei blocchi presenti questo blocco (this) con "ciao" come nome.

```
setCreativeTab(CreativeTabs.tabBlock);
```

indica in quali delle tabelle della modalità Creative sarà possibile trovare il blocco. Senza questo non possiamo trovare il blocco per testarlo nel gioco! Questa linea di codice dice che il blocco si trova nella tab dei blocchi.

```
setUnlocalizedName("ciao");
```

da il nome al blocco. E' necessario che questo nome sia uguale a quello di GameRegistry.registerBlock!

<u>Attenzione agli import:</u>
Spesso, abbiamo bisogno di indicare da dove vengono le funzioni che usiamo quando scriviamo determinate parole chiave. Possiamo usare nella nostra classe funzioni provenienti da altre classi se "importiamo" le classi che contengono la descrizione di cosa fa quella funzione. Di solito, gli import si scrivono all'inizio del file. Quelli utili per aggiungere il primo blocco sono:

```
import net.minecraft.block.Block;
import net.minecraft.block.material.Material;
import net.minecraft.block.state.IBlockState;
import net.minecraft.creativetab.CreativeTabs;
import net.minecraft.entity.Entity;
import net.minecraft.entity.player.EntityPlayer;
import net.minecraft.item.Item;
import net.minecraft.util.BlockPos;
import net.minecraft.util.EnumFacing;
import net.minecraft.world.World;
import net.minecraftforge.fml.common.registry.GameRegistry;
```

Come deve essere il file del BloccoEsempio alla fine del processo:

```
package com.example.examplemod;

import net.minecraft.block.Block;
//lista degli import...

public class BloccoEsempio extends Block{

    public BloccoEsempio(){
        super(Material.rock);
        GameRegistry.registerBlock(this, "ciao");
        setCreativeTab(CreativeTabs.tabBlock);
        setUnlocalizedName("ciao");
    }

}
```

Importante: il nuovo blocco va messo anche dentro ExampleMod.java, così:

```
public class ExampleMod{
        public static final String MODID = "examplemod";
        public static final String VERSION = "1.0";

        Block i;

        @EventHandler
        public void preinit(FMLPreInitializationEvent event){
         i = new BloccoEsempio();
        }

        @EventHandler
        public void init(FMLInitializationEvent event){

        }
}
```

Se ora apriamo il gioco e entriamo in un mondo qualsiasi in modalità Creative, possiamo trovare il nostro blocco:

E' lì, è viola e nero (perché non ha una texture associata!) e non è molto speciale: quello che abbiamo fatto fino ad ora è stato solo e soltanto costruire un nuovo blocco "generico".

importante:
ricorda il nome che hai dato al blocco (in questo caso, si chiama "ciao", perché ci tornerà utile in seguito)

## Aggiungere caratteristiche al proprio blocco:

All'interno del costruttore si possono aggiungere alcune caratteristiche per rendere il blocco speciale. Tutte queste proprietà devono essere seguite da parentesi tonde che contengono il valore che volete inserire.

> I numeri con la F: queste proprietà vogliono dentro le parentesi dei numeri con la F. In java questi numeri si chiamano Float, e in italiano si traduce con "virgola mobile". Significa, semplicemente, che sono numeri decimali, ma per renderlo chiaro in java scriviamo una F subito dopo il numero.

| | | |
|---|---|---|
| **setHardness** | Indica quanto ci vuole per rompere un blocco | default: dipende dal materiale, ad esempio stone ha 1.5F, obsidian ha 50.0F |
| **setResistance** | Indica la resistenza di un blocco alle esplosioni | default: dipende dal materiale: stone:10.0F, obsidian:2000.0F |
| **setStepSound** | Che suono fa il blocco quando lo si calpesta? | default: dipende dal materiale |
| **setLightLevel** | Quanta luce emette il blocco? | default: 0.0F (non emette luce), massimo: 1.0F |
| **setBlockUnbreakable** | Rende il blocco indistruttibile in survival | default: solo la bedrock è indistruttibile |

Esempio:

```
public BloccoEsempio(){
        super(Material.rock);
        GameRegistry.registerBlock(this, "ciao");
        setCreativeTab(CreativeTabs.tabBlock);
        setUnlocalizedName("ciao");
        setHardness(0.5F);
        setLightLevel(1F);
    }
```

## Blocchi che reagiscono alle azioni

Al di fuori del costruttore possiamo inserire una funzione aggiuntiva, che ci permette di far fare delle azioni ad un blocco nel caso in cui il blocco venga attivato. Questo è un esempio:

```
public boolean onBlockActivated(World w, BlockPos p, IBlockState s,
                EntityPlayer player, EnumFacing ss, float a, float b, float c) {
        player.fallDistance = 0;
        player.motionY += 2;
        return true;
    }
```

Ignoriamo, per ora, tutto quello che c'è scritto nelle prime due righe, perché sarà il gioco stesso a occuparsi di riempire tutti quei valori. Quello che fa questo blocco quando viene attivato è far saltare in alto il giocatore grazie alle due righe:

```
        player.fallDistance = 0;
        player.motionY += 2;
```

## Il codice di BloccoEsempio fino ad ora:

```
package com.example.examplemod;

import net.minecraft.block.Block;
//lista degli import...

public class BloccoEsempio extends Block{

    public BloccoEsempio(){
        super(Material.rock);
        GameRegistry.registerBlock(this, "ciao");
        setCreativeTab(CreativeTabs.tabBlock);
        setUnlocalizedName("ciao");
        setHardness(0.5F);
        setLightLevel(1F);
    }

    public boolean onBlockActivated(World w, BlockPos p, IBlockState s,
                EntityPlayer player, EnumFacing ss, float a, float b, float c) {
         player.fallDistance = 0;
         player.motionY += 2;
         return true;
    }

}
```

## Aggiungere una ricetta che dia come risultato il nuovo blocco

Torniamo su ExampleMod.java e alle nostre conoscenze in fatto di aggiunta di ricette.
Fare una ricetta che dia come risultato il nostro blocco è praticamente uguale al fare una ricetta che da come risultato un blocco qualsiasi: l'unica differenza è che dobbiamo mettere un riferimento al nostro blocco nel posto dove va messo il risultato della ricetta.

così, dentro ExampleMod:

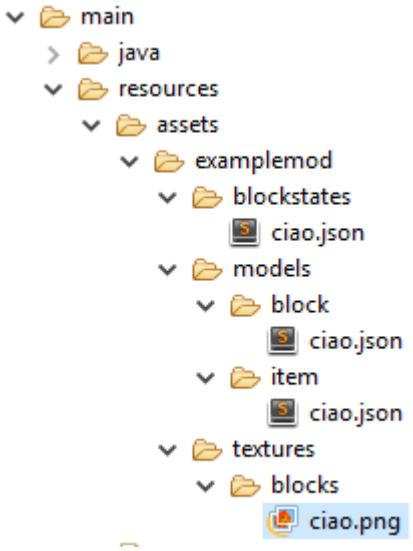
```
public void preinit(FMLPreInitializationEvent event){
         i = new BloccoEsempio();

         GameRegistry.addRecipe(new ItemStack(i), new Object[] {
            "AA",
            "AA",
            'A', Blocks.dirt});
}
```

# Aggiungere la texture al proprio blocco


Abbiamo sistemato il funzionamento del cubo, ma ora abbiamo bisogno di dargli alcune informazioni aggiuntive sul modello e assegnargli una texture.

Costruiamo una struttura di cartelle per contenere i nuovi file:
apriamo da esplora risorse la cartella di forge, dopodiché apriamo la cartella “src”, e poi la cartella “main”. Ignoriamo la cartella “java”. Da ora in poi, se le cartelle non ci sono, vanno create. Devono essere sistemate in questo modo:




Come si può vedere, i files all’interno delle cartelle “blockstates”, “models/block” e “models/item” sono i files corrispondenti al cubo appena creato. All’interno del gioco avevo scelto di dare come nome al mio cubo “ciao”, dunque i files che stiamo per creare dovranno avere lo stesso nome.

Per creare un file .json, si può cliccare col tasto destro del mouse sul fondo della cartella in cui ci troviamo, creare un nuovo documento qualsiasi, dopodiché cambiare l’estensione del documento in .json rinominando il file. Potete aprire un file json con qualsiasi editor di testo (anche con notepad).

Rinominare un file:
cliccare con il tasto destro sul file, dopodiché selezionare “rinomina”.

Una volta creati i file, li riempiamo con qualsiasi editor di testo.

**ciao.json dentro la cartella blockstates deve contenere:**

```
{
	"variants": {
	"normal": { "model": "examplemod:ciao" }
	}
}
```

**ciao.json dentro models/block deve contenere:**

```
{
	"parent": "block/cube_all",
	"textures": {
	"all": "examplemod:blocks/ciao"
	}
}
```

**ciao.json dentro models/item deve contenere:**

```
{
	"parent": "examplemod:block/ciao",
	"display": {
	"thirdperson": {
		"rotation": [ 10, -45, 170 ],
		"translation": [ 0, 1.5, -2.75 ],
		"scale": [ 0.375, 0.375, 0.375 ]
	}
}
```

**Ultima cosa: riprendiamo in mano ExampleMod.java e aggiungiamo le righe evidenziate in rosso dentro la funzione init:**

```
package com.example.examplemod;
import net.minecraft.block.Block;
//lista degli import...
@Mod(modid = ExampleMod.MODID, version = ExampleMod.VERSION)

public class ExampleMod{
	public static final String MODID = "examplemod";
	public static final String VERSION = "1.0";

	Block i;

	@EventHandler
	public void preinit(FMLPreInitializationEvent event){
	 i = new BloccoEsempio();
	}

	@EventHandler
	public void init(FMLInitializationEvent event){
	  if(event.getSide() == Side.CLIENT){
		Minecraft w = Minecraft.getMinecraft();
		RenderItem r = w.getRenderItem();
		ItemModelMesher m = r.getItemModelMesher();
		m.register(Item.getItemFromBlock(i), 0, new
ModelResourceLocation("examplemod:ciao", "inventory"));
	  }
	}

}
```

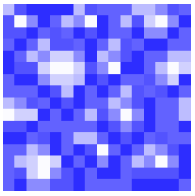
Attenzione: potete chiaramente vedere che in questi pezzi di codice, se li leggete, trovate più punti in cui c'è scritto "examplemod" e "ciao". Questi nomi si riferiscono alla mod che stiamo facendo ora, ma nel caso in cui voleste cambiare il nome della mod o il nome del cubo, sentitevi liberi di cambiarli, tenendo i nomi coerenti con quelli che avete usato nel resto del codice.

L'ultimo passo: aggiungere l'immagine. Deve avere:

- sempre lo stesso nome del cubo (dunque per me si chiama "ciao.png")
- essere quadrata
- essere della dimensione che volete
- essere un'immagine .png

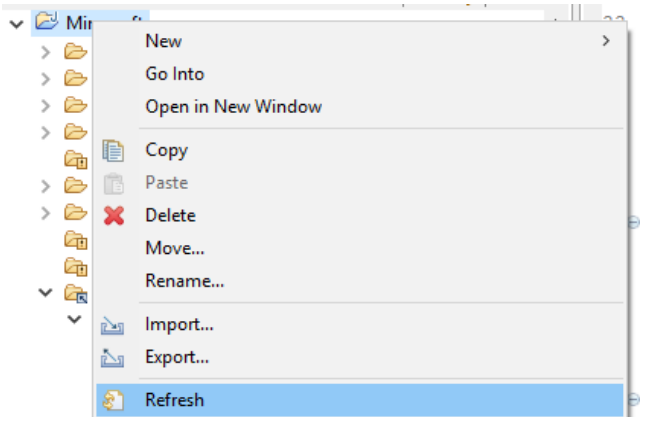
e va messa dentro la cartella textures/blocks.

Io, ad esempio, userò questa immagine:



*[Sorgente dell'immagine]*

Una volta sistemato anche quest'ultimo dettaglio, possiamo provare se funziona tutto.
Nel caso in cui le nuove cartelle non siano comparse in eclipse, possiamo fare un refresh per forzarlo a ricaricare tutti i file e a scoprire se ne abbiamo inseriti di nuovi nel progetto: clicchiamo con il destro sulla cartella principale del progetto ("Minecraft" o "MDKexample", insomma, quella che in eclipse sta più in alto di tutte) e selezioniamo refresh.




Ora testiamo tutto compilandolo ed eseguendolo:


ecco come è venuto il mio cubo :)

**SFIDA!**

- Vietato usare i propri computer!
- Squadre da 3!

link alle dispense: https://goo.gl/lbhGlz

10 punti fare un cubo personalizzato

Abbiamo visto come si fa un cubo personalizzato, ora dovete rifarne uno (standard) da soli!

10 punti aggiungere proprietà al cubo

Aggiungete delle proprietà come hardness e resistance al cubo!

10 punti aggiungere click col destro = salto

Aggiungete al cubo la funzione per saltare quando lo si aziona con il click destro.

10 punti fai una ricetta che dia come risultato il tuo nuovo cubo

Un piccolo indizio per fare questo punto:

Quando creiamo il nuovo blocco diciamo che i = new BloccoEsempio(), la stessa "i" potremmo metterla al posto di Block.diamonds in una ricetta…

Bonus per chi ha finito:

20 punti Aggiungi la texture al tuo cubo!

E' tutto scritto nelle dispense!

La texture più bella riceverà più punti!

Nella lezione di oggi aggiungeremo ancora più funzionalità al nostro blocco.
Per esempio, due propiètà molto importanti sono le seguenti:

- `setHardness(valore)`
  dove `valore` è un numero che indica la resistenza di un blocco. Per esempio, 1.5F è la resistenza della pietra, e 50.0F è la resistenza dell'ossidiana. (Attenzione: ricordarsi di aggiungere sempre un "F" subito dopo il numero)
- `setHarvestLevel(strumento, livello)`
  dove `strumento` indica che tipo di utensile dobbiamo usare per scavarlo (ad es. "pickaxe", "axe", "shovel"...), e invece `livello` è un numero che indica la qualità dello strumento da usare (0 = legno; 1 = pietra; 2 = ferro; 3 = diamante)

Se vogliamo che il nostro blocco faccia qualcosa quando viene attivato attraverso redstone, dobbiamo definire questa funzione nella classe BloccoEsempio.java:

```
public void onNeighborBlockChange(World w, BlockPos pos,
        IBlockState s, Block n) {
        if (w.isBlockPowered(pos) {
                Minecraft.getMinecraft().thePlayer.sendChatMessage("wow")
        }
}
```

In questo modo, ogni volta che il blocco viene attivato da redstone, scrive in chat "wow"; tuttavia, si può benissimo fargli scrivere dei comandi, e farlo comportare come un vero e proprio command block.

Inoltre, dovremo sapere bene alcuni comandi da chat (o da command block):

- /clear <player> - cancella tutto l'inventario
- /give <player> <item_id> <amount> - da un oggetto a un giocatore. L'item_id deve essere una parola del tipo "minecraft:planks"
- /kill <player> - ucccide un mostro/giocatore
- /particle <name> <x> <y> <z> <x2> <y2> <z2> - spawna particles di genere name (ad es. "explode", "bubble", "smoke") nel cubo definito dai punti alle coordinate x,y,z e x2,y2,z2
- /setblock <x> <y> <z> <TileName> - trasforma il blocco alle coordinate x,y,z nel blocco TileName (ad es. "minecraft:stone")
- /summon <entityName> - spawna un entity di nome entityName (ad es. "Bat", "WitherSkull", "PrimedTNT")
- /time set <value> - setta il tempo a value (ad es. "day", "night", "7000")
- /tp <player> <x> <y> <z> - teletrasporta il giocatore alle coordinate x,y,z
- /xp <value> - dona un numero uguale a value di punti esperienza al giocatore

*Sostituite i valori tra le "<>" con ciò che desiderate.*
*Inoltre il tag "<player>" può assumere diversi valori:*

- *Se lo lasciate vuoto, indica automaticamente il giocatore che esegue il comando*
- *Può essere l'username di un giocatore diverso (se si gioca in multiplayer)*
- *Se è "@p", indica il giocatore più vicino a dove viene eseguito il comando*
- *Se è "@r", viene indicato un giocatore a caso*
- *Se è "@a", il comando viene eseguito su tutti i giocatori presenti*
- *Se è "@e", il comando viene eseguito su tutte i giocatori e i mostri presenti*

*Avanzate:*
*Se al posto del tag player avete usato per esempio "@a", che seleziona tutti i giocatori, potete mettere subito dopo delle parentesi quadre in cui scrivete delle restrizioni. Ad esempio, scrivere @a[r=20] seleziona tutti i giocatori in un raggio di 20 blocchi da dove viene eseguito il comando, poichè la lettera "r" sta per raggio. Invece, scrivere @a[lm=5], seleziona tutti i giocatori di livello maggiore o uguale a 5, poichè "lm" sta per livello minimo. Un altro ancora: @a[m=1] seleziona tutti i giocatori in creative mode, poichè "m" sta per mode. Si possono mettere più restrizioni nelle parentesi quadre, basta separarle con una virgola. Esempio finale:*

```
/give @a[m=0,lm=10] minecraft:diamond_block 64
```

*Questo comando da 64 blocchi di diamante a tutti i giocatori in survival mode che abbiano almeno 10 livelli di esperienza.*

# SFIDA:

[10 pti] - Creare un blocco che ci mette esattamente 5 secondi ad essere scavato con un piccone di diamante

[10 pti] - Creare un bottone che scrive qualcosa in chat appena viene premuto

[20 pti] - Creare un bottone che spawna un Creeper appena viene premuto

[15 pti] - Creare un blocco che dona al giocatore una spada di diamante appena viene cliccato con il destro (bonus: 15 pti in più se la spada di diamante è enchantata)

[30 pti] - Creare un bottone che spawna un TNT accesa appena viene premuto

[50 pti] - Creare un bottone “negozio”, che dona a tutti i giocatori una spada di diamante solo se hanno un livello di esperienza maggiore o uguale a 3

[100 pti] - Creare un sistema di due leve in cui se vengono accese tutte e due vengono spawnate 125 TNT accese sparse