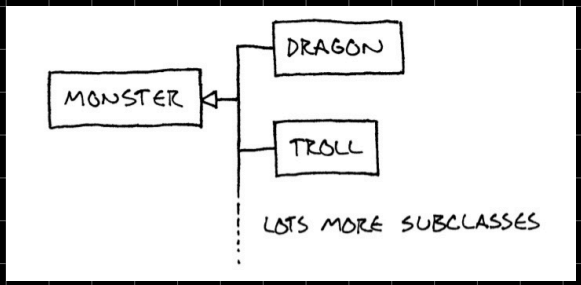
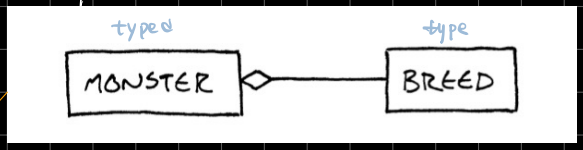
inheritance



Hard to tune stats

composition



↑ easier to bind data file

↑ easier to implement customized memory allocator

↓ hard to define behaviour

↓ loses class differences (ALL MONSTERS)

Simple breed + monster

```
class Breed
{
public:
  Breed(int health, const char* attack)
  : health_(health),
    attack_(attack)
  {}

  int getHealth() { return health_; }
  const char* getAttack() { return attack_; }

private:
  int health_; // Starting health.
  const char* attack_;
};
```

Very simple. It's basically just a container for two data fields: the starting health and the attack string. Let's see how monsters use it:

```
class Monster
{
public:
  Monster(Breed& breed)
  : health_(breed.getHealth()),
    breed_(breed)
  {}

  const char* getAttack()
  {
    return breed_.getAttack();
  }

private:
  // Current health.
  int health_;
  Breed& breed_;
};
```

- encapsulate type?
  - ↑ hidden complexity
  - ↑ easier to overwrite type behaviour
  - ↓ need lots of forwarding
  - ↑ support outside access <factory>
  - ↓ hard to manage
- how to create type
  - new object<&type>;
  - type.newObj();
  - control allocation outside
  - control allocation in type
- can type change?
  - ↑ simpler code
  - ↑ easier debug
  - ↑ less obj creation <new obj could be the old with new type>
  - ↓ careful when changing types

a factory pattern

```
class Breed
{
public:
  Monster* newMonster()
  {
    return new Monster(*this);
  }

  // Previous Breed code...
};
```

we can do fancier memory allocation here

LESS FLEXIBLE cannot change allocation outside

And the class that uses them:

```
class Monster
{
  friend class Breed;

public:
  const char* getAttack()
  {
    return breed_.getAttack();
  }

private:
  Monster(Breed& breed)
  : health_(breed.getHealth()),
    breed_(breed)
  {}

  int health_; // Current health.
  Breed& breed_;
};
```

do more behaviour specifically here

Easier to tune shared attributes

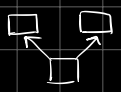
Breed with inheritance

```
class Breed
{
public:
  Breed(Breed* parent, int health,
        const char* attack)
  : parent_(parent),
    health_(health),
    attack_(attack)
  {}

  int         getHealth();
  const char* getAttack();

private:
  Breed*      parent_;
  int         health_; // Starting health.
  const char* attack_;
};
```

- data inheritance ?

no inheritance	single 	multiple 
↑ simple	↑ Still relatively simple	↑ avoid all duplication
↓ duplicate effort	↓ slower attribute lookup (go through inherit chains)	↓ complex