# Enter your codename from eClass grades as your name on Menti

# Did you enter your codename from eclass as your name on Mentimeter?

None of the options are correct!

No
Maybe
Yes

# Have you been employed as a programmer before?

✓
No
✓
Maybe
✓
Yes

# What programming languages do you know?

There's no correct answer!

# What do you think Software Engineering is?

There's no correct answer!

# Have you read the collaboration policy?

✓
✓
✓
No
Maybe
Yes

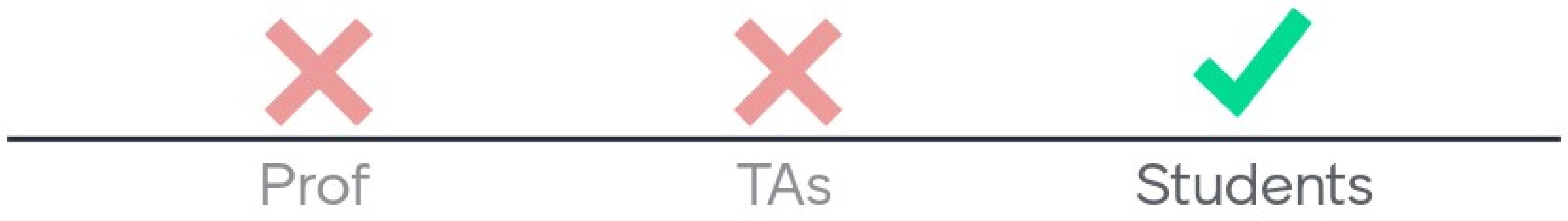
# Who is playing the role of the customer in 301?

| | | |
|---|---|---|
| ✓ | ✗ | ✗ |
| Prof | TAs | Students |

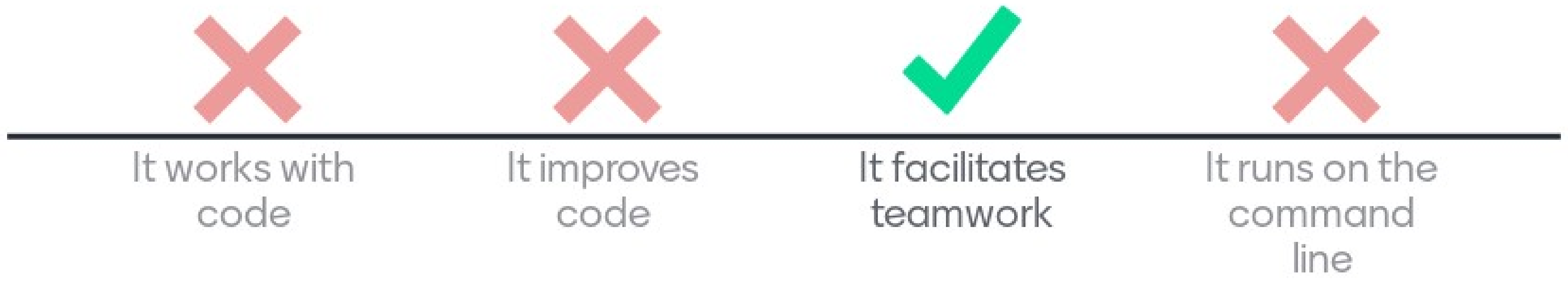
# Who is playing the role of the manager in 301?

| | | |
|---|---|---|
| ✗ | ✓ | ✗ |
| Prof | TAs | Students |

# Who is playing the role of the programmers in 301?



# why is git considered a software engineering tool?

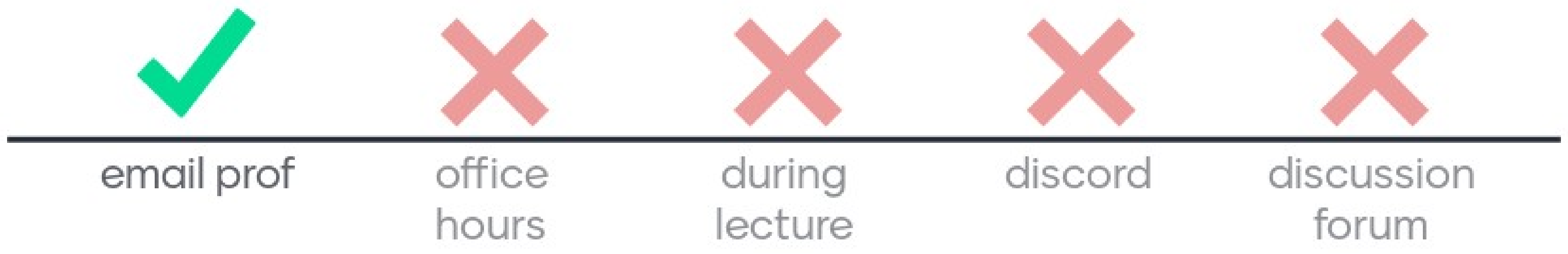


- ✗ It works with code
- ✗ It improves code
- ✓ It facilitates teamwork
- ✗ It runs on the command line

# If you have a question about the project requirements, where should you ask it?

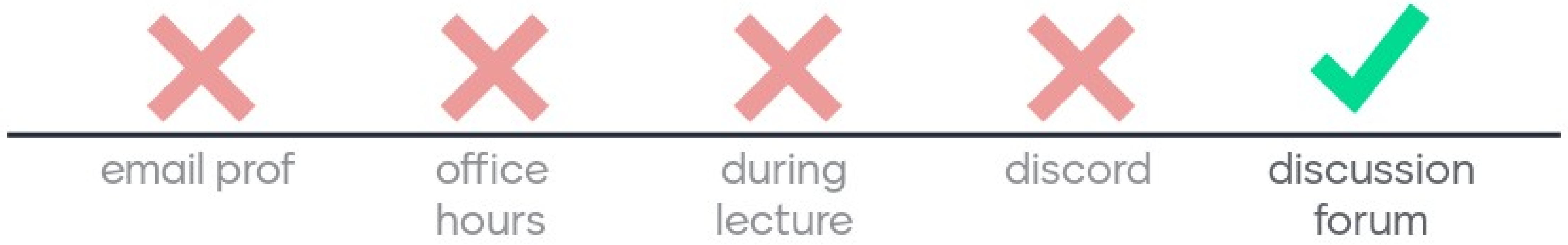
| | | | | |
|---|---|---|---|---|
| ✗ | ✗ | ✗ | ✗ | ✓ |
| email prof | office hours | during lecture | discord | discussion forum |

# If you think your grade is wrong, where should you ask about it?



| ✓ | ✗ | ✗ | ✗ | ✗ |
|---|---|---|---|---|
| email prof | office hours | during lecture | discord | discussion forum |

# If you have a question about an assignment requirement where should you ask it?



# If you need help with your code where should you ask about it?

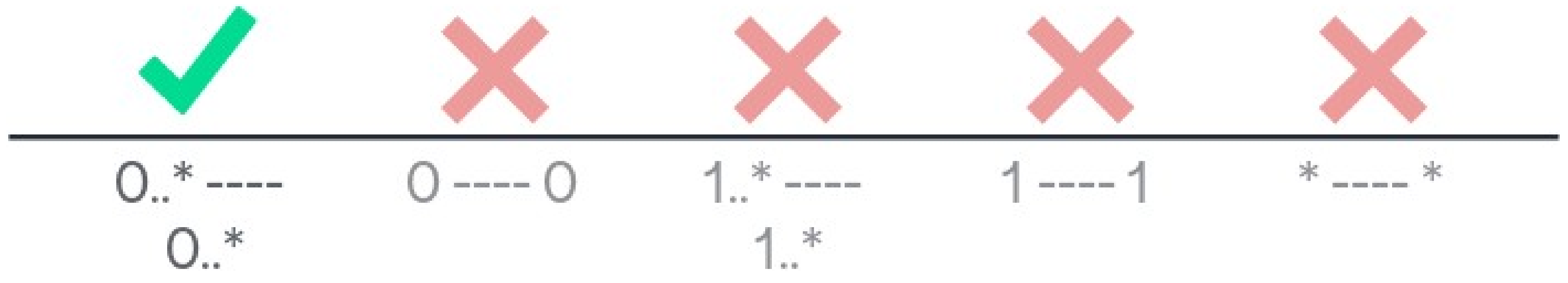
| email prof | office hours | during lecture | discord | discussion forum | lab |
|---|---|---|---|---|---|
| ✗ | ✓ | ✗ | ✗ | ✗ | ✓ |

# every Trainer object references zero or more Pokemon objects, every Pokemon object references zero or more Trainer objects

| ✓ | ✗ | ✗ |
|---|---|---|
| Association | Aggregation | Composition |

Mentimeter

# every Trainer object references zero or more Pokemon objects, every Pokemon object references zero or more Trainer objects



✓ 0..* ---- 0..*

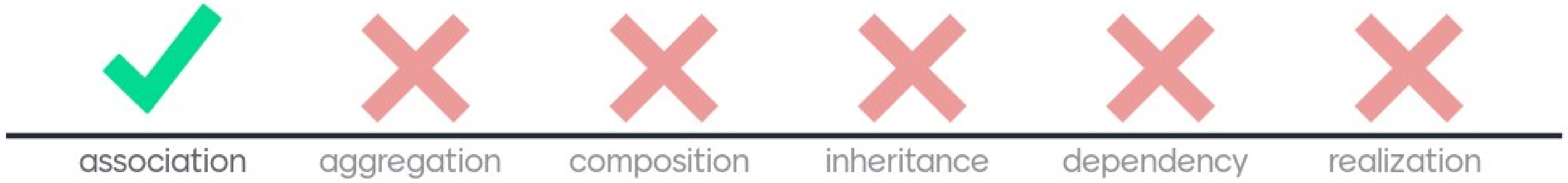
✗ 0 ---- 0

✗ 1..* ---- 1..*

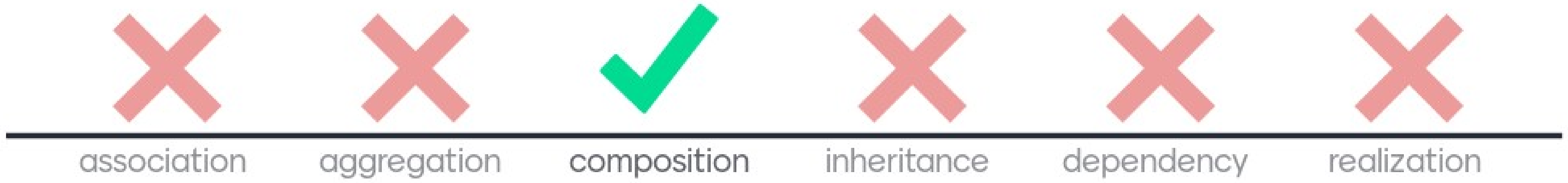
✗ 1 ---- 1

✗ * ---- *

# No diamond


✓
association
✗
aggregation
✗
composition
✗
inheritance
✗
dependency
✗
realization

# Filled-in diamond



✗ association

✗ aggregation

✓ composition

✗ inheritance

✗ dependency

✗ realization

# Outlined (empty) diamond

| ✗ | ✓ | ✗ | ✗ | ✗ | ✗ |
|---|---|---|---|---|---|
| association | aggregation | composition | inheritance | dependency | realization |

# If a Team is made out of one or more Pokemon, what is that?

| ✗ | ✗ | ✓ |
|---|---|---|
| Association | Aggregation | Composition |

# If a Pokemon has one or more Moves, what is that?

| ✗ | ✓ | ✗ |
|---|---|---|
| Association | Aggregation | Composition |

# If a Pokemon has one or more Moves, which is the Whole

✓ Pokemon is the Whole

✗ Pokemon is the Parts

# Which side does the diamond go on

- ✓ The whole
- ✗ The parts
- ✗ The middle

# What is NOT a reason to generalize?

| ✗ | ✗ | ✓ | ✗ |
|---|---|---|---|
| reuse code | reduce code | reduce abstractions | help us think about what we're even doing |

# Base class is also called a

| ✓ | ✗ | ✗ |
|---|---|---|
| superclass | subclass | interface |

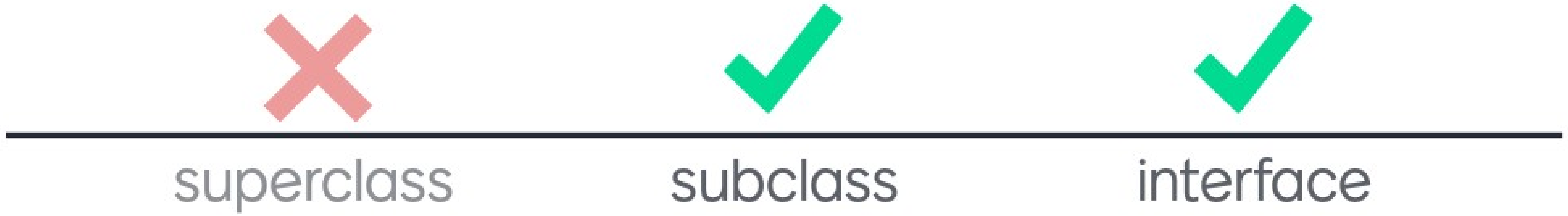
# Derived class is also called a

| ✗ | ✓ | ✗ |
|---|---|---|
| superclass | subclass | interface |

# a JAVA class can only have ONE

| ✓ | ✗ | ✗ |
|---|---|---|
| superclass | subclass | interface |

# a Java class can have multiple



✗ superclass

✓ subclass

✓ interface

# A is a superclass of B

| ✗ | ✓ | ✗ | ✗ |
|---|---|---|---|
| A extends B | B extends A | A implements B | B implements A |

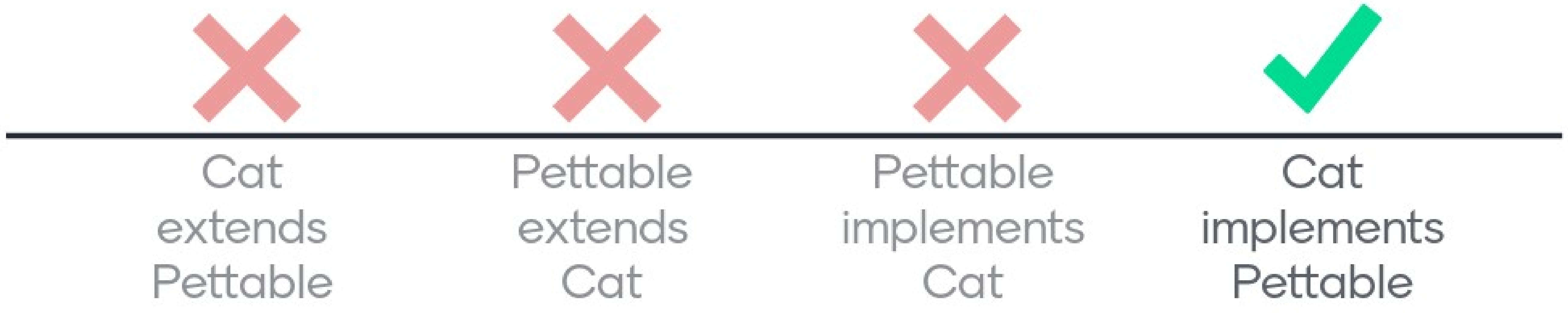
# A defines all the methods declared in B

| | | | |
|---|---|---|---|
| ✗ | ✗ | ✓ | ✗ |
| A extends B | B extends A | A implements B | B implements A |

# Which makes the most sense?

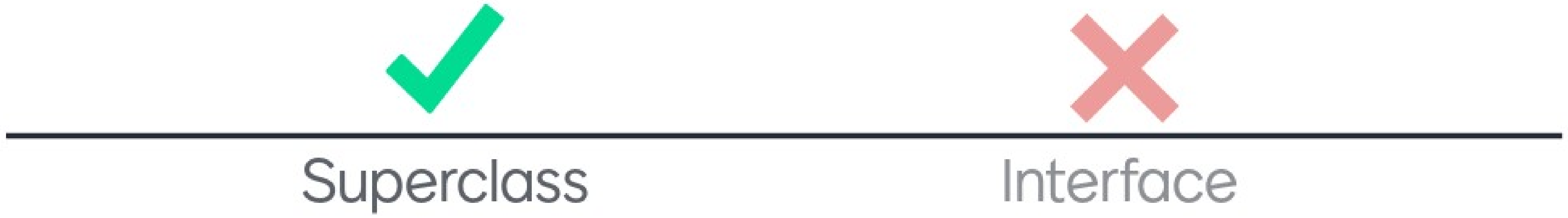
| ✗ | ✗ | ✓ | ✗ | ✗ | ✗ |
|---|---|---|---|---|---|
| Cat extends Dog | Dog extends Cat | Cat extends Animal | Animal extends Cat | Cat implements Animal | Animal implements Cat |

# Which makes the most sense?

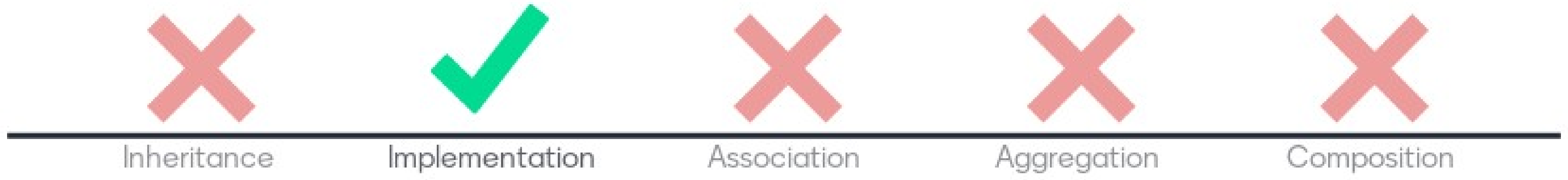


- ✗ Cat extends Pettable
- ✗ Pettable extends Cat
- ✗ Pettable implements Cat
- ✓ Cat implements Pettable

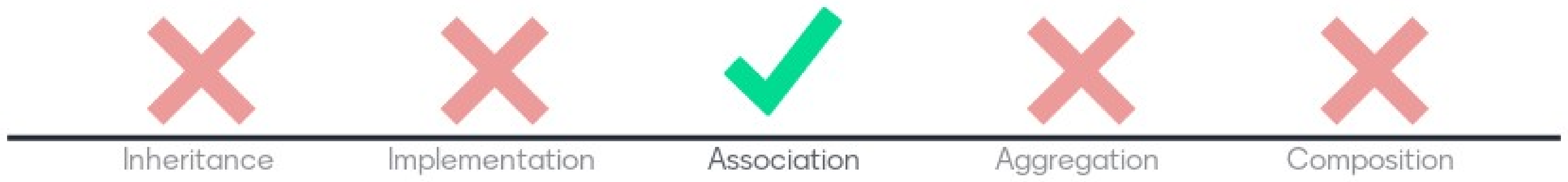
# Which one can have actual code in it?



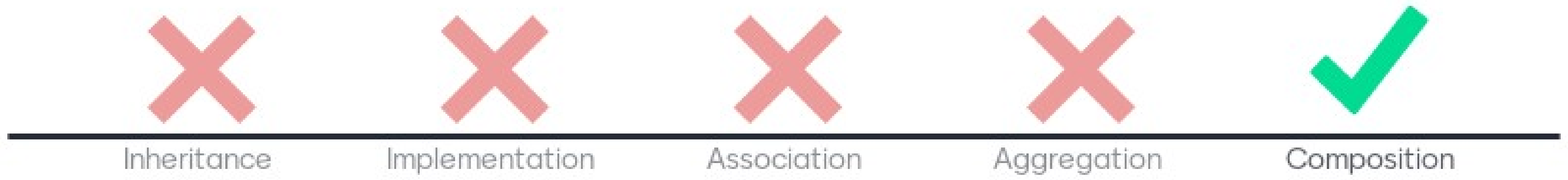
# What relationship do InBattleUsable and Potion have?



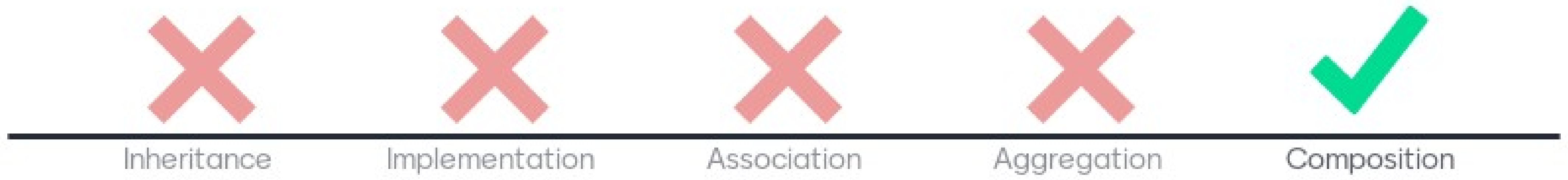
# What relationship do Character and Inventory have?



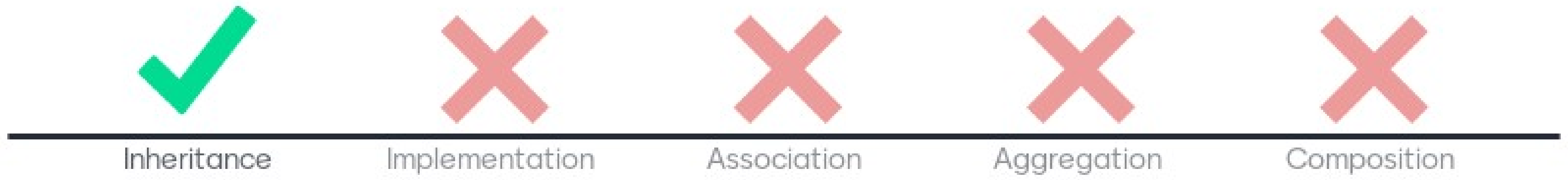
# What relationship do Inventory and Item have?



# What relationship do Inventory and Item have?



# What relationship do Item and Potion have?



# What visibility does inventory in character have?

✓ Private

✗ Protected

✗ Default

✗ Public

# What visibility does getInventory() in character have?

| | | | |
|---|---|---|---|
| ✗ | ✗ | ✗ | ✓ |
| Private | Protected | Default | Public |

# Overriding is...

✓ Same name, same parameters, but in a subclass

✗ Same name, same parameters, but in a superclass

✗ Same name, different parameters

# Overloading is...

| ✗ | ✗ | ✓ |
|---|---|---|
| Same name, same parameters, but in a subclass | Same name, same parameters, but in a superclass | Same name, different parameters |

# Upcast

| ✗ | ✓ | ✗ |
|---|---|---|
| Always safe, must be done explicitly | Always safe, can be done implicitly | Not always safe, could throw an error |

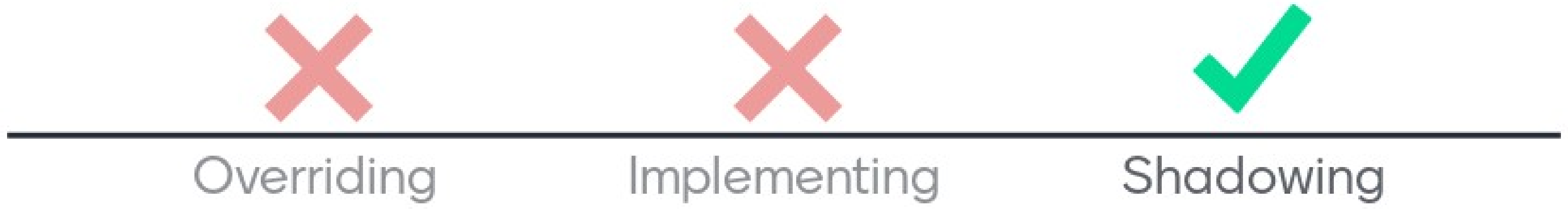
# Downcast

- ✗ Always safe, must be done explicitly
- ✗ Always safe, can be done implicitly
- ✓ Not always safe, could throw an error

# Downcast

| ✓ | ✗ |
|---|---|
| Must be done explicitly | Can be done implicitly |

# Attribute with the same name in a subclass as one that already exists in the superclass



✗ Overriding

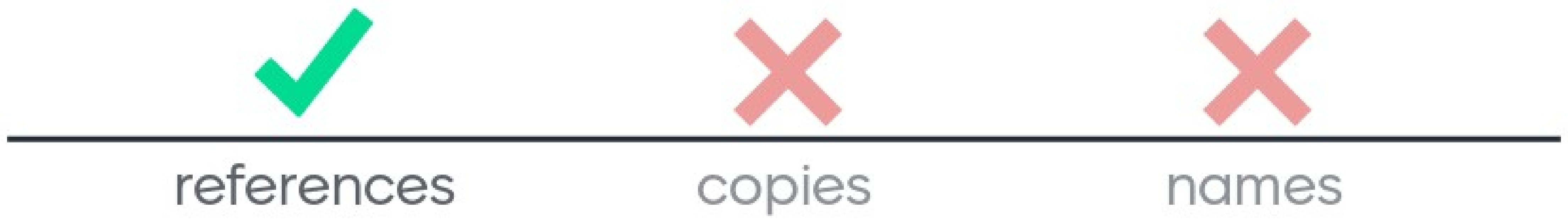
✗ Implementing

✓ Shadowing

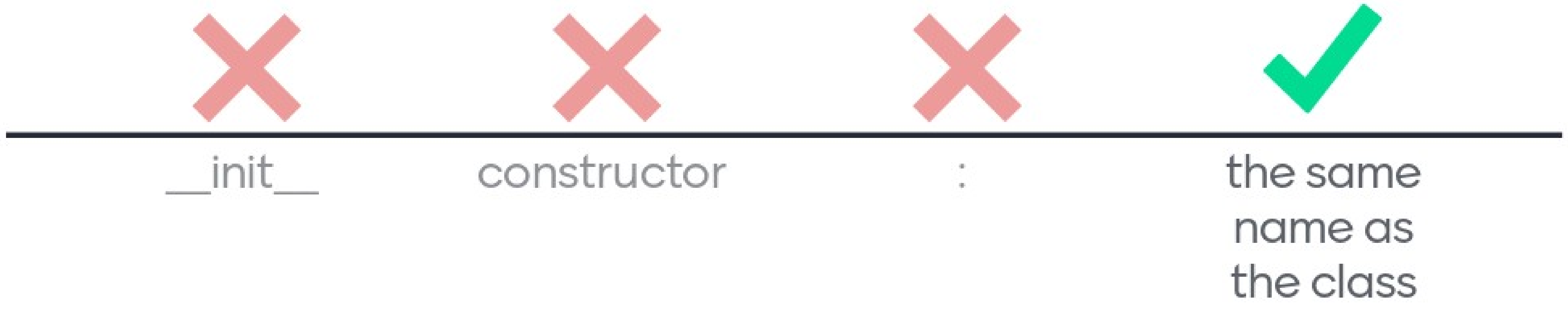
# In Java when I call a method, the arguments are passed

- ✗ by reference
- ✓ by value
- ✗ by name

# Even though its call-by-value, the values passed are usually



# Java constructors are named



__init__

constructor

:

the same name as the class

# Abstract classes provide

✗ no implementation

✗ full implementation

✓ partial implementation

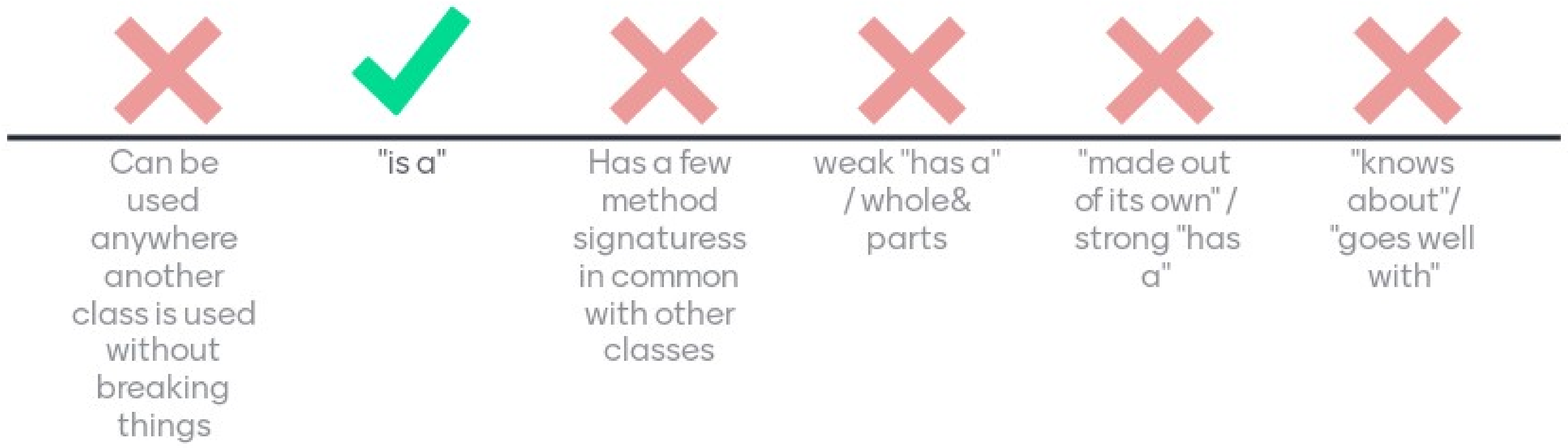
# Concrete (normal) classes provide

| ✗ | ✓ | ✗ |
|---|---|---|
| no implementation | full implementation | partial implementation |

# Interfaces provide

| | | |
|---|---|---|
| ✓ | ✗ | ✗ |
| no implementation | full implementation | partial implementation |

# What's the quick rule of thumb to determine if something should be an inheritance?

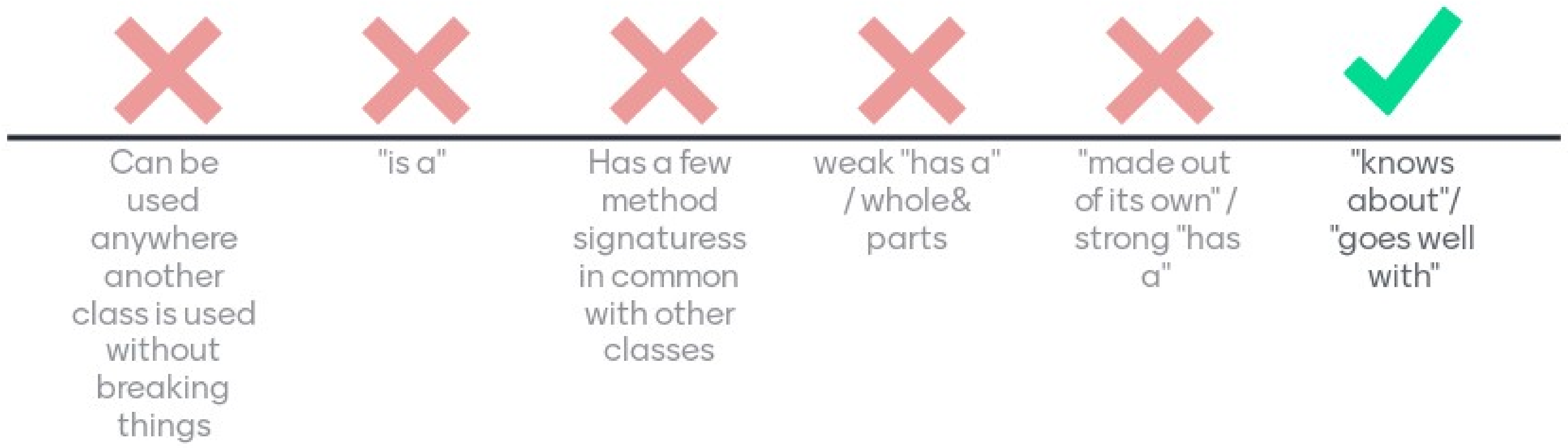


- ✗ Can be used anywhere another class is used without breaking things
- ✓ "is a"
- ✗ Has a few method signaturess in common with other classes
- ✗ weak "has a" / whole& parts
- ✗ "made out of its own" / strong "has a"
- ✗ "knows about"/ "goes well with"

# What's the Liskov Substitution principle to determine if something should be an inheritance?

| ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
|---|---|---|---|---|---|
| Can be used anywhere another class is used without breaking things | "is a" | Has a few method signaturess in common with other classes | weak "has a" / whole& parts | "made out of its own" / strong "has a" | "knows about"/ "goes well with" |

# What's the quick rule of thumb to determine if something should be an association?

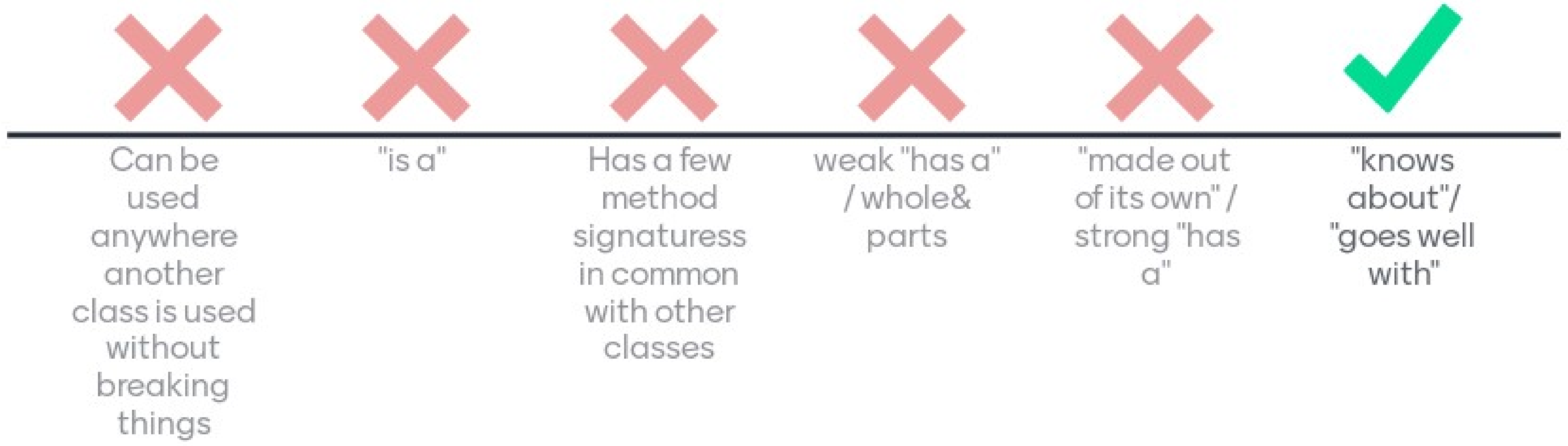


- ✗ Can be used anywhere another class is used without breaking things
- ✗ "is a"
- ✗ Has a few method signaturess in common with other classes
- ✗ weak "has a" / whole& parts
- ✗ "made out of its own" / strong "has a"
- ✓ "knows about"/ "goes well with"

# What's the quick rule of thumb to determine if something should be an aggregation?

| | | | | | |
|---|---|---|---|---|---|
| ✗ | ✗ | ✗ | ✓ | ✗ | ✗ |
| Can be used anywhere another class is used without breaking things | "is a" | Has a few method signaturess in common with other classes | weak "has a" / whole& parts | "made out of its own" / strong "has a" | "knows about"/ "goes well with" |

# What's the quick rule of thumb to determine if something should be an composition?



- ✗ Can be used anywhere another class is used without breaking things
- ✗ "is a"
- ✗ Has a few method signaturess in common with other classes
- ✗ weak "has a" / whole& parts
- ✗ "made out of its own" / strong "has a"
- ✓ "knows about"/ "goes well with"

# Is composition relationship enforced in Java?

| ✗ | ✓ | ✗ |
|---|---|---|
| No | Sometimes | Always |

# For a composition relationship...

| ✓ | ✗ | ✗ |
|---|---|---|
| Part instances are deleted when the Whole instance is | Doesn't have a Whole and Part(s) style relationship | Part instances can be shared (they aren't exclusive) |

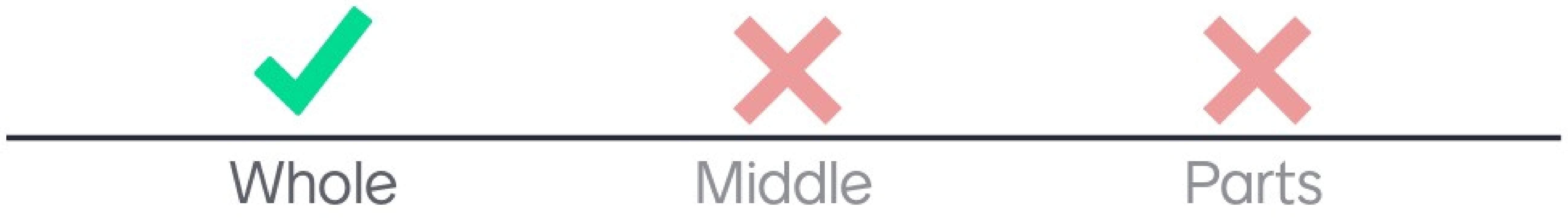
# For an aggregation relationship...

| ✗ | ✗ | ✓ |
|---|---|---|
| Part instances are deleted when the Whole instance is | Doesn't have a Whole and Part(s) style relationship | Part instances can be shared (they aren't exclusive) |

# For an association relationship...

| ✗ | ✓ | ✗ |
|---|---|---|
| Part instances are deleted when the Whole instance is | Doesn't have a Whole and Part(s) style relationship | Part instances can be shared (they aren't exclusive) |

# The diamond goes on the side of the ...

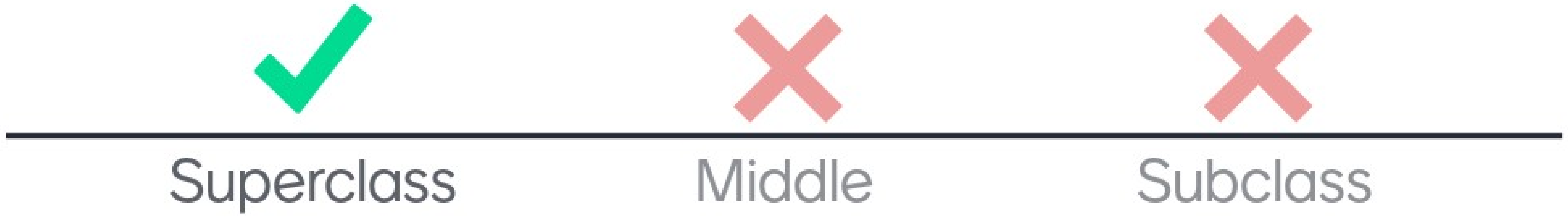


✓ Whole

✗ Middle

✗ Parts

# The arrow goes on the side of the ...



# The arrow goes on the side of the ...

| | | |
|---|---|---|
| ✓ | ✗ | ✗ |
| Interface | Middle | Class that `implements` interface |

# end