1. В PHP оператор === (строгое сравнение) проверяет не только значения, но и идентичность объектов, то есть, указывают ли переменные на один и тот же объект.

В данном случае создаются два разных объекта класса ClassA, следовательно, $a и $b ссылаются на разные экземпляры.

Поэтому результатом будет bool(false).

2. Оператор === для массивов проверяет совпадение ключей, значений и их порядка.

Оба массива полностью идентичны (ключи, значения и их порядок одинаковы).

Следовательно, результатом будет bool(true).

3. Определяется класс ClassA с публичным свойством var1 и конструктором, принимающим int $var1.

Создается объект $a с var1 = 5.

Затем переменной $b присваивается ссылка на объект $a.

Важно! В PHP объекты передаются по ссылке, а не копируются.

Изменяется значение $b->var1 = 10;, но так как $b и $a указывают на один и тот же объект, то $a->var1 тоже становится 10.

В итоге echo $a->var1; выведет 10.

4. Создается класс ClassA с публичным свойством var1 и конструктором.

Объект $a создается с var1 = 5.

Переменной $b присваивается ссылка на тот же объект, что и $a (в PHP объекты передаются по ссылке).

Устанавливаем $a->var1 = 10;. Так как $a и $b указывают на один и тот же объект, то var1 внутри объекта меняется на 10.

$a = null; — теперь $a больше не ссылается на объект, но сам объект все еще существует, потому что на него продолжает ссылаться $b.

echo $b->var1; — так как $b все еще указывает на объект, значение var1 останется 10.

5. Создается класс ClassA с публичным свойством var1 и конструктором.

Создается объект $a и его свойство var1 устанавливается в 5.

Создается переменная $b, которая является ссылкой на $a ($b = &$a;).

Теперь $a и $b ссылаются на один и тот же объект в памяти.

Меняется $b->var1 = 10;, что означает, что var1 внутри объекта теперь равно 10.

Присваивается $a = null;.

Так как $b является ссылкой на $a, а не просто копией, то $b тоже становится null.

Попытка выполнить echo $b->var1; вызовет ошибку, потому что $b теперь не объект, а null, и у null нет свойства var1.

6. Создается массив:

foreach ($numbers as $number) создает копию каждого элемента массива в переменной $number.

Это не изменяет оригинальный массив, так как $number — это локальная копия текущего элемента.

Оператор \*\*= 2 возводит каждую копию $number в квадрат, но не изменяет оригинальный массив.

В итоге, echo $numbers[4]; выводит оригинальное значение пятого элемента массива, то есть 5.

7. Создается массив:

$numbers = [1, 2, 3, 4, 5];

foreach ($numbers as &$number) передает элементы массива по ссылке:

1 \*\* 2 = 1 → $numbers = [1, 2, 3, 4, 5]

2 \*\* 2 = 4 → $numbers = [1, 4, 3, 4, 5]

3 \*\* 2 = 9 → $numbers = [1, 4, 9, 4, 5]

4 \*\* 2 = 16 → $numbers = [1, 4, 9, 16, 5]

5 \*\* 2 = 25 → $numbers = [1, 4, 9, 16, 25]

Теперь массив выглядит так:

[1, 4, 9, 16, 25]

Однако, важно помнить, что $number остается ссылкой на последний элемент массива ($numbers[4]).

Присваивается $number = 10;

Так как $number все еще ссылается на $numbers[4], изменяется сам массив:

[1, 4, 9, 16, 10]

echo $numbers[4]; выведет 10.

8. Объявляются два интерфейса:

interface Movable {

public function move(): void;

}

interface Breathable {

public function breath(): void;

}

Movable содержит метод move().

Breathable содержит метод breath().

Объявляется класс Animal, который реализует оба интерфейса:

class Animal implements Movable, Breathable {

public function move(): void {

echo self::class . ' move';

}

}

Класс Animal обязан реализовать move() (из Movable) и breath() (из Breathable).

Однако, метод breath() отсутствует! Это приведет к фатальной ошибке.

9. Объявляется интерфейс Movable, содержащий метод move().

Объявляется интерфейс Breathable, содержащий метод breath().

Объявляется интерфейс Dog, который наследует (extends) оба интерфейса:

Интерфейсы могут наследовать друг друга без ограничений.

Dog теперь объединяет методы из Movable, Breathable и добавляет bark().

Код выполнится успешно, так как в PHP интерфейсы могут наследовать другие интерфейсы.

10. Dog move Animal move

11. Объявляются интерфейсы Movable и Breathable

Movable требует метод move().

Breathable требует метод breath().

Класс Animal реализует оба интерфейса

Реализуется метод move(), но он в коде не используется.

Реализуется метод breath(), который выводит:

echo self::class . ' breath';

Использование self::class означает, что всегда будет выведено имя класса в котором определен метод (Animal).

Класс Dog наследует Animal, но не переопределяет методы

Dog получает методы move() и breath() от Animal.

Создается объект Dog, вызывается breath()

$dog = new Dog();

$dog->breath();

Метод breath() определен в Animal и использует self::class, а self всегда относится к классу, где метод определен, т.е. Animal.

Animal breath

Создается объект Animal, вызывается breath()

$animal = new Animal();

$animal->breath();

Здесь self::class также Animal, так что результат будет таким же:

Animal breath

12. Установит все зависимости согласно composer.lock (если есть) в vendor директорию, иначе использует composer.json.

Команда composer install устанавливает зависимости, указанные в файле composer.lock, если он существует. Если же composer.lock нет, то будет использован файл composer.json, и создан новый файл composer.lock с зафиксированными версиями пакетов.

13. Установит все зависимости согласно composer.json (даже если есть composer.lock файл) в vendor директорию, после обновит composer.lock.

Команда composer update обновляет все зависимости, указанные в composer.json, и, если пакеты были обновлены, обновляет файл composer.lock, чтобы зафиксировать новые версии. Она игнорирует текущий composer.lock файл и устанавливает зависимости согласно актуальному содержимому composer.json

14. **composer require <имя пакета>**

Команда composer require <имя пакета> установит новый пакет и автоматически добавит его в файл composer.json как зависимость.

15. 1.2.X

Сегмент X в версии композер пакета обычно обозначает патч-версии, которые отвечают за исправление ошибок и багов, без добавления нового функционала или изменения интерфейса. Эти версии сохраняют обратную совместимость с предыдущими, то есть не меняют интерфейс и не вносят функциональные изменения.

16.

SELECT

a.id AS article\_id,

a.title AS article\_title,

a.body AS article\_body,

c.id AS comment\_id,

c.title AS comment\_title,

c.body AS comment\_body,

c.parent\_id AS comment\_parent\_id

FROM articles a

JOIN comments c ON a.id = c.article\_id

WHERE a.id = 101;

17.

SELECT

u.id AS user\_id,

u.email,

u.first\_name,

u.last\_name,

u.photo\_link,

r.title AS role\_title

FROM users u

JOIN user\_roles ur ON u.id = ur.user\_id

JOIN roles r ON ur.role\_id = r.id

WHERE u.id = 256;

18.

INSERT INTO user\_roles (user\_id, role\_id)

VALUES (225, 8);

19.

UPDATE users

SET photo\_link = 'новая\_ссылка\_на\_фото'

WHERE id = 67;

20.

DELETE FROM user\_roles WHERE user\_id = 78;

DELETE FROM comments WHERE user\_id = 78;

DELETE FROM articles WHERE user\_id = 78;

DELETE FROM users WHERE id = 78;