Not Quite My Type

Using types to make impossible states truly impossible

Kai Sassnowski · @warsh33p · Laracon Online 2022

Hi, I'm Kai 👋

Website
kai-sassnowski.com

GitHub
github.com/ksassnowski

Twitter
@warsh33p

Soundcloud
N/A

Things I like
(in no particular order)

- **Static Analysis** tools (PHPStan, Psaml)
- **Well typed** code
- Having **tools** yell at me when I’m making mistakes
- My wife

Let’s look at some code

Preventing time travel

```php
class SystemClock implements ClockInterface
{
    public function wait(int $seconds): void
    {
        // ...
    }
}
```

```php
class SystemClock implements ClockInterface
{
    public function wait(int $seconds): void
    {
        // ...
    }
}
```

```
class SystemClock implements ClockInterface
{
    public function wait(69): void ✔
    {
        // ...
    }
}
```

```
class SystemClock implements ClockInterface
{
    public function wait(0): void  ✔
    {
        // ...
    }
}
```

Time travel, baby!

```
class SystemClock implements ClockInterface
{
    public function wait(-5): void
    {
        // ...
    }
}
```

The Problem

- This is a **perfectly valid** use of the method
- It's also **complete nonsense**
- But PHP and static analysers **don't know** this

```
class SystemClock implements ClockInterface
{
    public function wait(-5): void
    {
        // ...
    }
}
```

```php
class SystemClock implements ClockInterface
{
    public function wait(int $seconds): void
    {
        // ...
    }
}
```

```php
class SystemClock implements ClockInterface
{
    public function wait(int $seconds): void
    {
        if ($seconds < 0) {
            throw new InvalidArgumentException();
        }

        // ...
    }
}
```

$seconds isn’t an integer

Integers can be negative, $seconds can't

$seconds is a **duration**

Let’s narrow our types!

```php
final class Duration
{
    public function __construct(
        public readonly int $seconds
    ) {
    }
}
```

```php
final class Duration
{
    public function __construct(
        public readonly int $seconds
    ) {
    }
}
```

```php
final class Duration
{
    public function __construct(
        public readonly int $seconds
    ) {
    }
}
```

```php
final class Duration
{
    public function __construct(
        public readonly int $seconds
    ) {
        if ($seconds < 0) {
            throw new InvalidArgumentException();
        }
    }
}
```

```php
final class Duration
{
    public function __construct(
        public readonly int $seconds
    ) {
        if ($seconds < 0) {
            throw new InvalidArgumentException();
        }
    }
}
```

```php
final class Duration
{
    public function __construct(
        public readonly int $seconds
    ) {
        if ($seconds < 0) {
            throw new InvalidArgumentException();
        }
    }

    public static function seconds(int $seconds): self
    {
        return new self($seconds);
    }
}
```

```php
class SystemClock implements ClockInterface
{
    public function wait(int $seconds): void
    {
        if ($seconds < 0) {
            throw new InvalidArgumentException();
        }
    }
}
```

```
class SystemClock implements ClockInterface
{
    public function wait(Duration $duration): void
    {
        if ($seconds < 0) {
            throw new InvalidArgumentException();
        }
    }
}
```

```php
class SystemClock implements ClockInterface
{
    public function wait(Duration $duration): void
    {
        if ($seconds < 0) {
            throw new InvalidArgumentException();
        }
    }
}
```

```php
class SystemClock implements ClockInterface
{
    public function wait(Duration $duration): void
    {
        // …
    }
}
```

```php
$clock->wait(Duration::seconds(5));
```

✔ Works as expected

```php
$clock->wait(-5);
```

✔ Type Error: Expected `Duration`, got `int` instead

```php
$clock->wait(Duration::seconds(-5));
```

✔ InvalidArgumentException: Duration cannot be negative

Noice 👍

Seller-friendly discounts

```php
class ShoppingCart
{
    public function __construct(
        private array $products,
    ) {
    }

    public function total(): int {}
}
```

```
class ShoppingCart
{
    public function __construct(
        private array $products
    ) {
    }

    public function total(): int {}
}
```

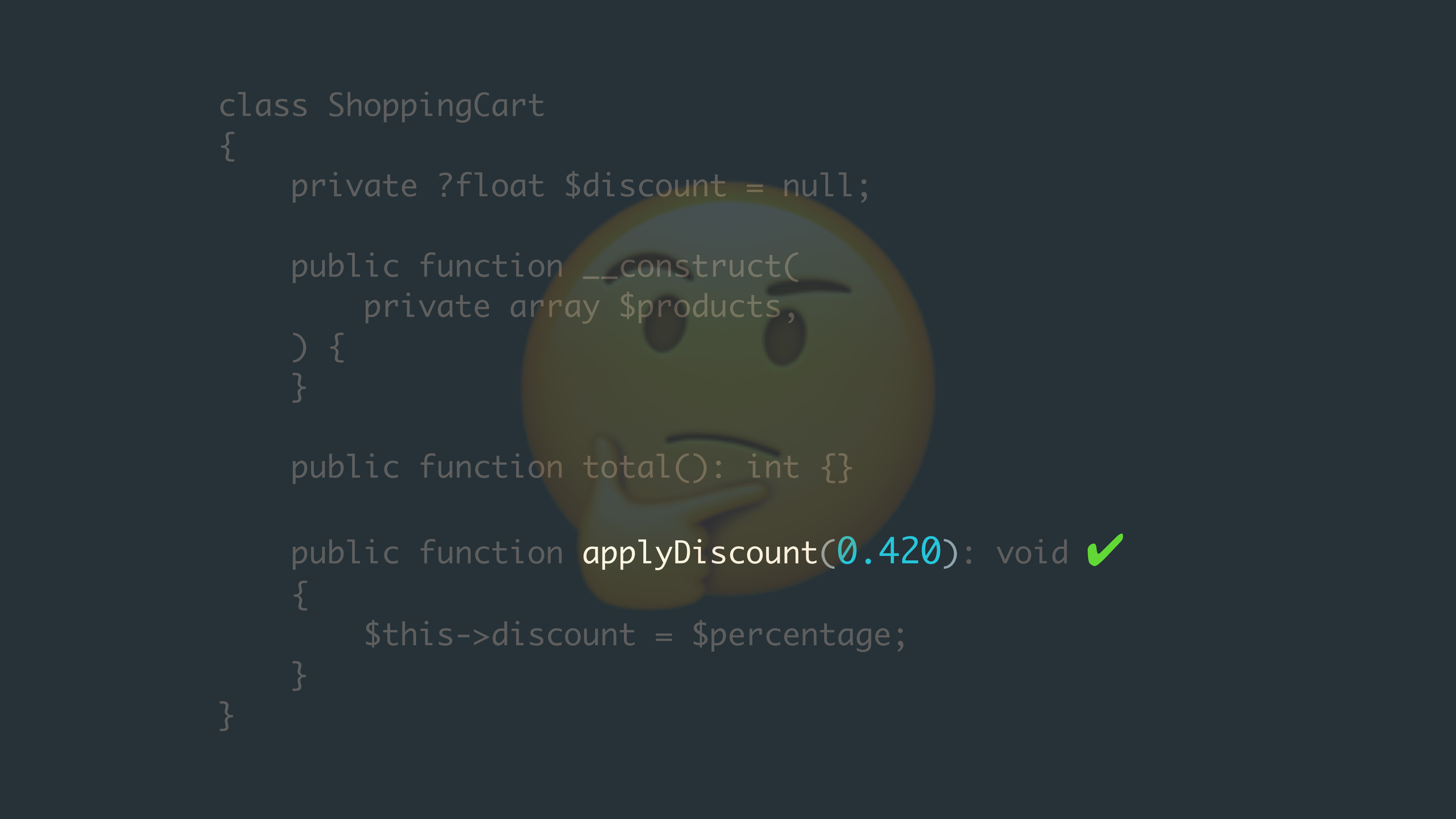
```php
class ShoppingCart
{
    public function __construct(
        private array $products,
    ) {
    }

    public function total(): int {}
}
```

Let’s save some money!

```php
class ShoppingCart
{
    private ?float $discount = null;

    public function __construct(
        private array $products,
    ) {
    }

    public function total(): int {}

    public function applyDiscount(float $percentage): void
    {
        $this->discount = $percentage;
    }
}
```

```php
class ShoppingCart
{
    private ?float $discount = null;

    public function __construct(
        private array $products,
    ) {
    }

    public function total(): int {}

    public function applyDiscount(float $percentage): void
    {
        $this->discount = $percentage;
    }
}
```

```php
class ShoppingCart
{
    private ?float $discount = null;

    public function __construct(
        private array $products,
    ) {
    }

    public function total(): int {}

    public function applyDiscount(float $percentage): void
    {
        $this->discount = $percentage;
    }
}
```


class ShoppingCart
{
private ?float $discount = null;
public function __construct(
private array $products,
) {
}
public function total(): int {}
public function applyDiscount(0.420): void ✔
{
$this->discount = $percentage;
}
}

```php
class ShoppingCart
{
    private ?float $discount = null;

    public function __construct(
        private array $products,
    ) {
    }

    public function total(): int {}

    public function applyDiscount(0.0): void
    {
        $this->discount = $percentage;
    }
}
```

```php
class ShoppingCart
{
    private ?float $discount = null;

    public function __construct(
        private array $products,
    ) {
    }

    public function total(): int {}

    public function applyDiscount(-0.1): void
    {
        $this->discount = $percentage;
    }
}
```

```php
class ShoppingCart
{
    private ?float $discount = null;

    public function __construct(
        private array $products,
    ) {
    }

    public function total(): int {}

    public function applyDiscount(2.0): void
    {
        $this->discount = $percentage;
    }
}
```

The Problem

- This is a **perfectly valid** use of the method
- It's also **complete nonsense**
- But PHP and static analysers **don't know** this

```
class ShoppingCart
{
    private ?float $discount = null;

    public function __construct(
        private array $products,
    ) {
    }

    public function total(): int {}

    public function applyDiscount(2.0): void
    {
        $this->discount = $percentage;
    }
}
```

```php
class ShoppingCart
{
    private ?float $discount = null;

    public function __construct(
        private array $products,
    ) {
    }

    public function total(): int {}

    public function applyDiscount(float $percentage): void
    {
        $this->discount = $percentage;
    }
}
```

```php
class ShoppingCart
{
    private ?float $discount = null;

    public function __construct(
        private array $products,
    ) {
    }

    public function total(): int {}

    public function applyDiscount(float $percentage): void
    {
        if ($percentage <= 0.0 || $percentage > 1.0) {
            throw new InvalidArgumentException();
        }

        $this->discount = $percentage;
    }
}
```

A discount isn’t a float

Floats can be negative, a discount can’t

Floats can be zero, a discount can’t

Floats can be greater than 1, a discount can’t

A discount is a **Discount**

Let’s narrow our types!

```php
final class Discount
{
    public function __construct(
        public readonly float $percentage,
    ) {
    }
}
```

```php
final class Discount
{
    public function __construct(
        public readonly float $percentage,
    ) {
    }
}
```

```php
final class Discount
{
    public function __construct(
        public readonly float $percentage,
    ) {
    }
}
```

```php
final class Discount
{
    public function __construct(
        public readonly float $percentage,
    ) {
        if ($percentage <= 0 || $percentage > 1.0) {
            throw new InvalidArgumentException();
        }
    }
}
```

```php
final class Discount
{
    public function __construct(
        public readonly float $percentage,
    ) {
        if ($percentage <= 0 || $percentage > 1.0) {
            throw new InvalidArgumentException();
        }
    }
}
```

```php
final class Discount
{
    public function __construct(
        public readonly float $percentage
    ) {
        if ($percentage <= 0 || $percentage > 1.0) {
            throw new InvalidArgumentException();
        }
    }

    public static function percentOff(int $value): self
    {
        return new self($value / 100);
    }
}
```

```php
final class Discount
{
    public function __construct(
        public readonly float $percentage
    ) {
        if ($percentage <= 0 || $percentage > 1.0) {
            throw new InvalidArgumentException();
        }
    }

    public static function percentOff(int $value): self
    {
        return new self($value / 100);
    }
}
```

```php
class ShoppingCart
{
    private ?float $percentage = null;

    public function __construct(
        private array $products,
    ) {
    }

    public function total(): int {}

    public function applyDiscount(float $percentage): void
    {
        if ($percentage <= 0.0 || $percentage > 1.0) {
            throw new InvalidArgumentException();
        }

        $this->discount = $discount;
    }
}
```

```php
class ShoppingCart
{
    private ?Discount $discount = null;

    public function __construct(
        private array $products,
    ) {
    }

    public function total(): int {}

    public function applyDiscount(Discount $discount): void
    {
        if ($percentage <= 0.0 || $percentage > 1.0) {
            throw new InvalidArgumentException();
        }

        $this->discount = $discount;
    }
}
```

```php
class ShoppingCart
{
    private ?Discount $discount = null;

    public function __construct(
        private array $products,
    ) {
    }

    public function total(): int {}

    public function applyDiscount(Discount $discount): void
    {
        if ($percentage <= 0.0 || $percentage > 1.0) {
            throw new InvalidArgumentException();
        }

        $this->discount = $discount;
    }
}
```

```php
class ShoppingCart
{
    private ?Discount $discount = null;

    public function __construct(
        private array $products,
    ) {
    }

    public function total(): int {}

    public function applyDiscount(Discount $discount): void
    {
        $this->discount = $discount;
    }
}
```

```php
$cart->applyDiscount(Discount::percentOff(25));
```

✔ Works as expected

```php
$cart->applyDiscount(-0.1);
```

✔ Type Error: Expected `Discount`, got `float` instead

```php
$cart->applyDiscount(Discount::percentOff(-10));
```

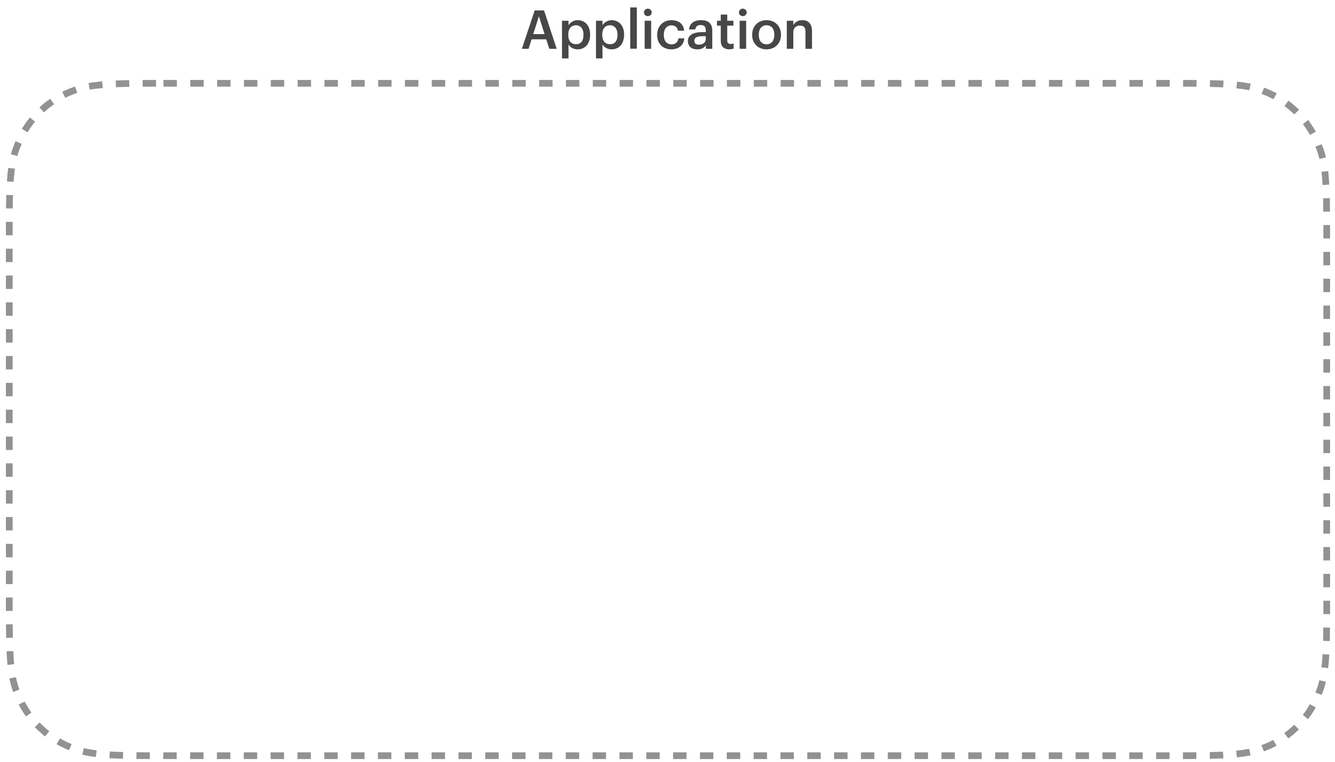
✔ InvalidArgumentException: Discount cannot be negative

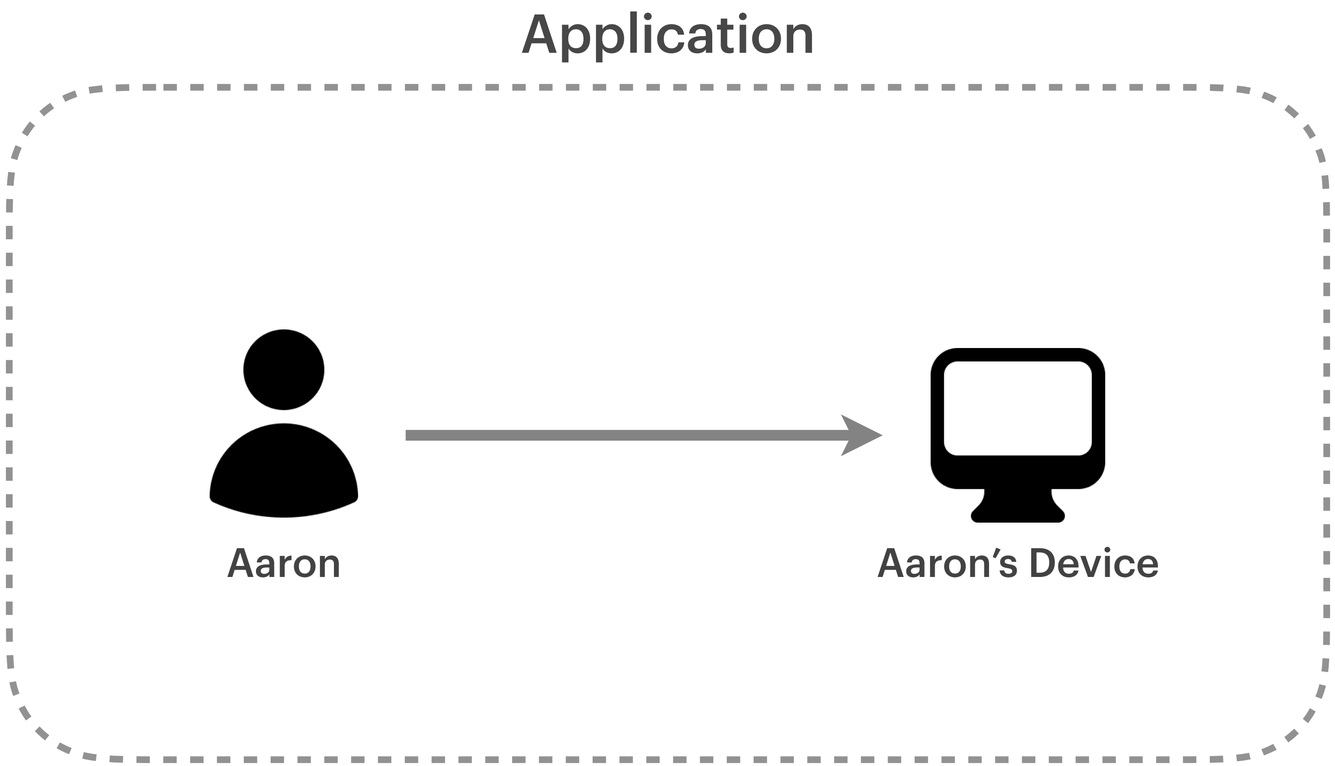
```php
$cart->applyDiscount(Discount::percentOff(200));
```

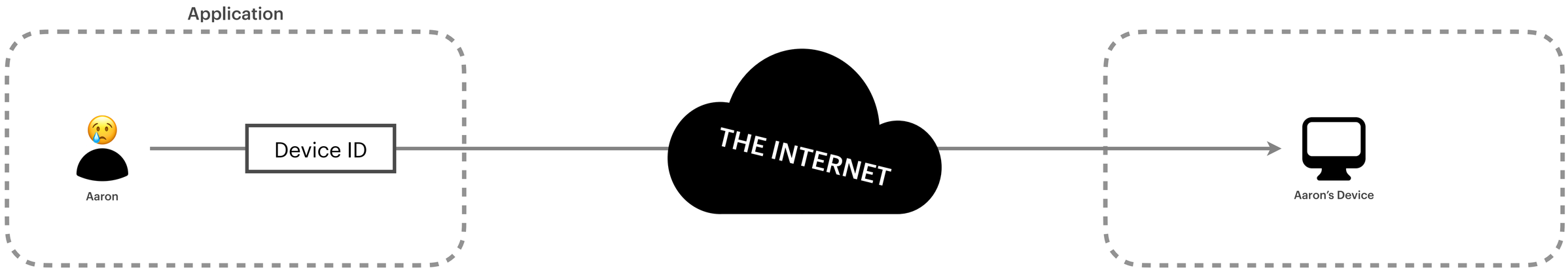
✔ InvalidArgumentException: Discount cannot be greater than 100

Noice 👍

Device IDs


Application


Application
Aaron
Aaron's Device


Application
Aaron
Device ID
THE INTERNET
Aaron's Device

```php
/**
 * @property string $device_id
 */
class User extends Model
{
}
```

```php
/**
 * @property string $device_id
 */
class User extends Model
{
}
```

```php
/**
 * @property string $device_id
 */
class User extends Model
{
}
```

```php
class DeviceRepository
{
    public function find(string $deviceID): ?Device
    {
        // …
    }
}
```

```php
class DeviceRepository
{
    public function find(string $deviceID): ?Device
    {
        // …
    }
}
```

```php
class DeviceRepository
{
    public function find(string $deviceID): ?Device
    {
        // …
    }
}
```

```php
class DeviceRepository
{
    public function find($user->device_id): ?Device
    {
        // …
    }
}
```

✔

```
class DeviceRepository
{
    public function find(""): ?Device
    {
        // …
    }
}
```

```
class DeviceRepository
{
    public function find("not a uuid"): ?Device
    {
        // …
    }
}
```

```
class DeviceRepository
{
    public function find(string $deviceID): ?Device
    {
        // …
    }
}
```

```php
class DeviceRepository
{
    public function find(string $deviceID): ?Device
    {
        if (! \Ramsey\Uuid\Uuid::isValid($deviceID)) {
            return null;
        }

        // …
    }
}
```

A device ID isn’t a string

Strings can be empty, a device ID can’t

Strings don’t have to be valid UUIDs, a device ID does

A device ID is a **UUID**

Let’s narrow our types!

```php
/**
 * @property string $device_id
 */
class User extends Model
{
}
```

```php
/**
 * @property UuidInterface $device_id
 */
class User extends Model
{
}
```

```php
/**
 * @property UuidInterface $device_id
 */
class User extends Model
{
}
```

```php
/**
 * @property UuidInterface $device_id
 */
class User extends Model
{
    protected $casts = [
    ];
}
```

See: https://laravel.com/docs/eloquent-mutators#value-object-casting

```php
/**
 * @property UuidInterface $device_id
 */
class User extends Model
{
    protected $casts = [
        'device_id' => ???,
    ];
}
```

See: https://laravel.com/docs/eloquent-mutators#value-object-casting

```php
/**
 * @property UuidInterface $device_id
 */
class User extends Model
{
    protected $casts = [
        'device_id' => UuidCast::class,
    ];
}
```

See: https://laravel.com/docs/eloquent-mutators#value-object-casting

```php
/**
 * @property UuidInterface $device_id
 */
class User extends Model
{
    protected $casts = [
        'device_id' => UuidCast::class,
    ];
}
```

```php
class DeviceRepository
{
    public function find(string $deviceID): ?Device
    {
        if (! \Ramsey\Uuid\Uuid::isValid($deviceID)) {
            return null;
        }

        // …
    }
}
```

```
class DeviceRepository
{
    public function find(UuidInterface $deviceID): ?Device
    {
        if (! \Ramsey\Uuid\Uuid::isValid($deviceID)) {
            return null;
        }

        // …
    }
}
```

```php
class DeviceRepository
{
    public function find(UuidInterface $deviceID): ?Device
    {
        if (! \Ramsey\Uuid\Uuid::isValid($deviceID)) {
            return null;
        }

        // …
    }
}
```

```php
class DeviceRepository
{
    public function find(UuidInterface $deviceID): ?Device
    {
        // …
    }
}
```

```php
$repository->find($user->device_id);
```

✔ Works as expected

```php
$repository->find("");
```

✔ Type Error: Expected UuidInterface, got string instead

Oh by the way, users have UUIDs too

```php
/**
 * @property UuidInterface $device_id
 */
class User extends Model
{
    protected $casts = [
        'device_id' => UuidCast::class,
    ];
}
```

```php
/**
 * @property UuidInterface $uuid
 * @property UuidInterface $device_id
 */
class User extends Model
{
    protected $casts = [
        'uuid' => UuidCast::class,
        'device_id' => UuidCast::class,
    ];
}
```

```php
$repository->find($user->device_id);
```

✔ Works as expected

```php
$repository->find("");
```

✔ Type Error: Expected UuidInterface, got string instead

```php
$repository->find($user->uuid);
```

You smug bastard

A device ID isn’t a UUID

A device ID is a **DeviceID**

Let’s narrow our types

```php
class DeviceID
{
    public function __construct(
        public readonly string $uuid,
    ) {
    }
}
```

```php
class DeviceID
{
    public function __construct(
        public readonly string $uuid,
    ) {
    }
}
```

```php
class DeviceID
{
    public function __construct(
        public readonly string $uuid,
    ) {
    }
}
```

```php
class DeviceID
{
    public function __construct(
        public readonly string $uuid,
    ) {
        if (! \Ramsey\Uuid\Uuid::isValid($uuid)) {
            throw new InvalidArgumentException();
        }
    }
}
```

```php
class DeviceID
{
    public function __construct(
        public readonly string $uuid,
    ) {
        if (! \Ramsey\Uuid\Uuid::isValid($uuid)) {
            throw new InvalidArgumentException();
        }
    }
}
```

```php
/**
 * @property UuidInterface $uuid
 * @property UuidInterface $device_id
 */
class User extends Model
{
    protected $casts = [
        'uuid' => UuidCast::class,
        'device_id' => UuidCast::class,
    ];
}
```

See: https://laravel.com/docs/eloquent-mutators#value-object-casting

```php
/**
 * @property UuidInterface $uuid
 * @property DeviceID $device_id
 */
class User extends Model
{
    protected $casts = [
        'uuid' => UuidCast::class,
        'device_id' => DeviceIDCast::class,
    ];
}
```

See: https://laravel.com/docs/eloquent-mutators#value-object-casting

```php
/**
 * @property UuidInterface $uuid
 * @property DeviceID $device_id
 */
class User extends Model
{
    protected $casts = [
        'uuid' => UuidCast::class,
        'device_id' => DeviceIDCast::class,
    ];
}
```

See: https://laravel.com/docs/eloquent-mutators#value-object-casting

```php
class DeviceRepository
{
    public function find(UuidInterface $deviceID): ?Device
    {
        // …
    }
}
```

```
class DeviceRepository
{
    public function find(DeviceID $deviceID): ?Device
    {
        // …
    }
}
```

```php
class DeviceRepository
{
    public function find(DeviceID $deviceID): ?Device
    {
        // …
    }
}
```

```
$repository->find($user->device_id);
✔ Works as expected


$repository->find("");
✔ Type Error: Expected UuidInterface, got string instead


$repository->find($user->uuid);
✔ Type Error: Expected DeviceID, got UuidInterface instead
```

Noice 👍

Nice over-engineering, bro

```php
class SystemClock implements ClockInterface
{
    public function wait(int $seconds): void
    {
        if ($seconds < 0) {
            throw new InvalidArgumentException();
        }
    }
}
```

```php
class ShoppingCart
{
    public function applyDiscount(float $percentage): void
    {
        if ($percentage <= 0.0 || $percentage > 1.0) {
            throw new InvalidArgumentException();
        }
    }
}
```

```php
class DeviceRepository
{
    public function find(string $deviceID): ?Device
    {
        if (! \Ramsey\Uuid\Uuid::isValid($deviceID)) {
            return null;
        }
    }
}
```

```php
public function wait(int $seconds): void
{
    if ($seconds < 0) {
        throw new InvalidArgumentException();
    }
}
```

```php
public function applyDiscount(float $percentage): void
{
    if ($percentage <= 0.0 || $percentage > 1.0) {
        throw new InvalidArgumentException();
    }
}
```

```php
public function find(string $deviceID): ?Device
{
    if (! \Ramsey\Uuid\Uuid::isValid($deviceID)) {
        return null;
    }
}
```

Any integer is fine!

```
public function wait(int $seconds): void
{
    if ($seconds < 0) {
        throw new InvalidArgumentException();
    }
}
```

Any float is fine!

```
public function applyDiscount(float $percentage): void
{
    if ($percentage <= 0.0 || $percentage > 1.0) {
        throw new InvalidArgumentException();
    }
}
```

Any string is fine!

```
public function find(string $deviceID): ?Device
{
    if (! \Ramsey\Uuid\Uuid::isValid($deviceID)) {
        return null;
    }
}
```

```php
public function wait(int $seconds): void
{
    if ($seconds < 0) {
        throw new InvalidArgumentException();
    }
}
```

```php
public function applyDiscount(float $percentage): void
{
    if ($percentage <= 0.0 || $percentage > 1.0) {
        throw new InvalidArgumentException();
    }
}
```

```php
public function find(string $deviceID): ?Device
{
    if (! \Ramsey\Uuid\Uuid::isValid($deviceID)) {
        return null;
    }
}
```

just kidding, lol

If a parameter has to satisfy certain invariants, *consider* **promoting it to an object that guarantees these invariants**

Thanks!

Not Quite My Type

Using types to make impossible states truly impossible

Kai Sassnowski · @warsh33p · Laracon Online 2022