

# Not Quite My Type

Using types to make impossible states truly impossible

# Not Quite My Type

Using types to make impossible states truly impossible

# Hi, I'm Kai



## **Website**

kai-sassnowski.com

## **GitHub**

github.com/ksassnowski

## **Twitter**

@warsh33p

## **Soundcloud**

N/A



# Thinks I like

(in no particular order)

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

**Let's look at some code**

**Preventing time travel**

```
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
    {
        // ...
    }
}
```

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

```
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!**

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

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

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

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

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

```
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);
    }
}
```



```
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();
        }
    }
}
```

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

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

✓ Works as expected

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

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

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

✓ `InvalidArgumentException`: Duration cannot be negative

Noice 👍

# **Seller-friendly discounts**

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

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



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

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

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

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

**Let's save some money!**

```
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(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(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;
    }
}
```

```
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;
    }
}
```



```
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;
    }
}
```

```
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;
    }
}
```

```
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(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!**

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

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

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

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

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



```
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);
    }
}
```

```
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);
    }
}
```

```
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;
    }
}
```



```
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;
    }
}
```

```
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;
    }
}
```

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

✓ Works as expected

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

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

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

✓ InvalidArgumentException: Discount cannot be negative

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

✓ InvalidArgumentException: Discount cannot be greater than 100

Noice 👍

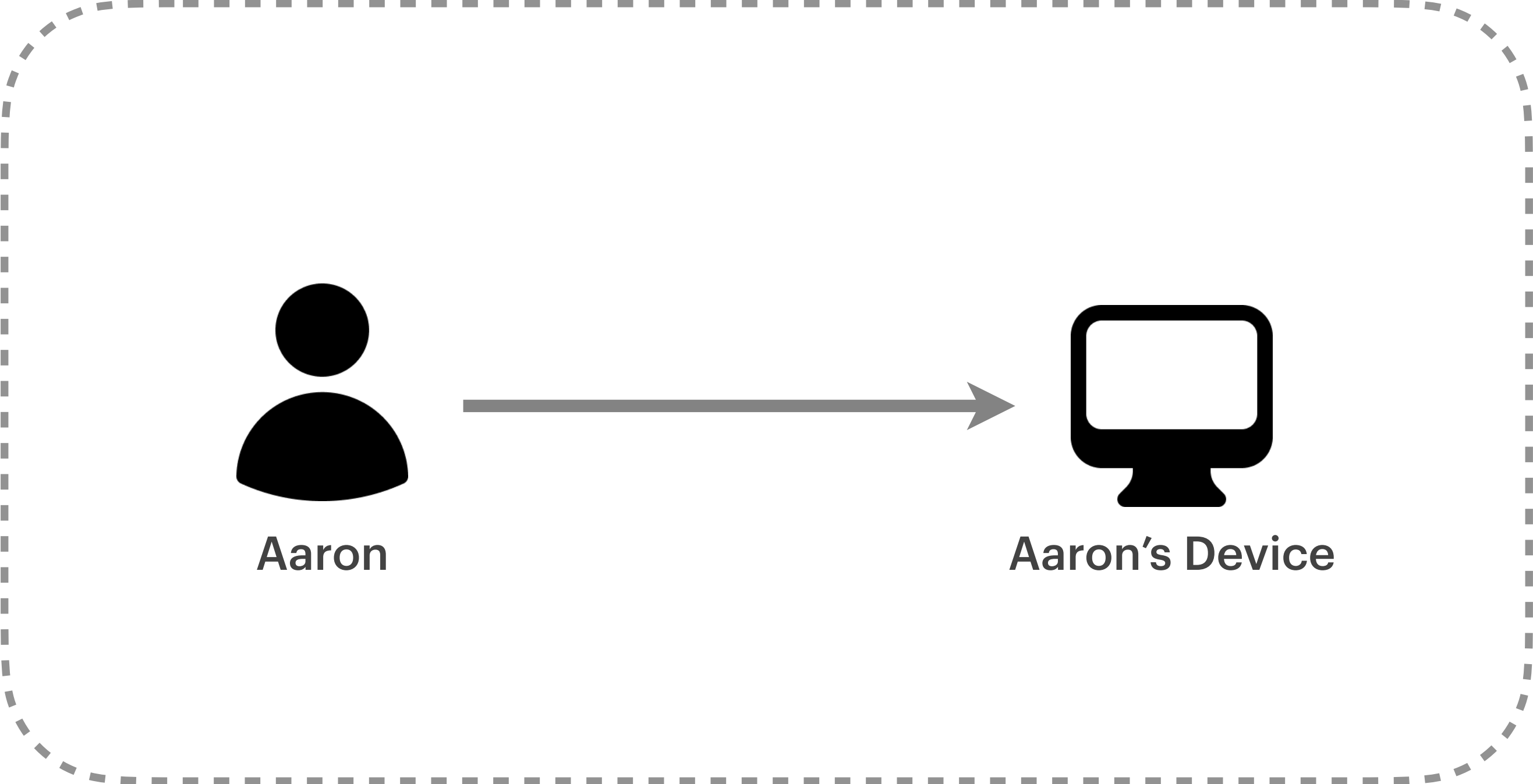


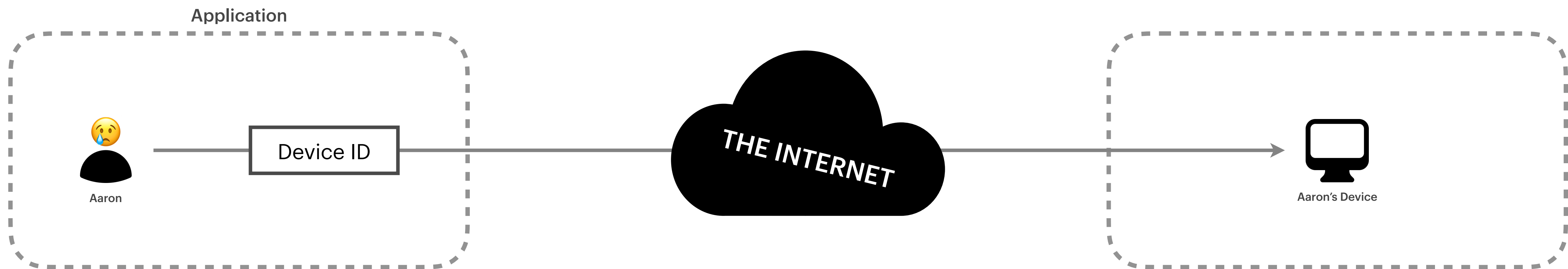
# Device IDs

Application



Application











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





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

```
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 $deviceId): ?Device
    {
        if (! \Ramsey\Uuid\Uuid::isValid($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!**











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

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

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

```
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;
        }

        // ...
    }
}
```

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

        // ...
    }
}
```

```
class DeviceRepository
{
    public function find(UuidInterface $deviceId): ?Device
    {
        // ...
    }
}
```

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

✓ Works as expected

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

✓ Type Error: Expected UuidInterface, got string instead



Oh by the way, users have UUIDs too

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

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

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

✓ Works as expected

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

✓ Type Error: Expected UuidInterface, got string instead

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

You smug bastard

**A device ID isn't a UUID**

A device ID is a **DeviceID**

Let's narrow our types

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



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

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

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

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

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

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

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

```
class DeviceRepository
{
    public function find(UuidInterface $deviceId): ?Device
    {
        // ...
    }
}
```



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

```
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);
```

✓ YouTypeError: Expected DeviceID, got UuidInterface instead

Noice 👍

**Nice over-engineering, bro**

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

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

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

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

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

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





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

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

```
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