

Generic Structures | *impl* Blocks

■ Implementing Functionality

- ◆ Generic implementation
 - Implements functionality for *any* type that can be used with the structure
- ◆ Concrete implementation
 - Implements functionality for only the type specified

Concrete Implementation – Setup

```
trait Game {  
    fn name(&self) -> String;  
}
```

```
enum BoardGame {  
    Chess,  
    Monopoly,  
}
```

```
enum VideoGame {  
    PlayStation,  
    Xbox,  
}
```

```
impl Game for BoardGame { ...  
}
```

```
impl Game for VideoGame { ...  
}
```

■ Concrete Implementation – Usage

```
struct PlayRoom<T: Game> {  
    game: T,  
}
```

```
impl PlayRoom<BoardGame> {  
    pub fn cleanup(&self) { ...  
    }  
}
```

```
let video_room = PlayRoom {  
    game: VideoGame::Xbox,  
};
```

```
let board_room = PlayRoom {  
    game: BoardGame::Monopoly,  
};  
board_room.cleanup();  
video_room.cleanup();
```

■ Concrete Implementation – Error

```
19 | struct PlayRoom<T: Game> {  
   | ----- method `cleanup` not found for this  
...  
58 |     video_room.cleanup();  
   |           ^^^^^^^^ method not found in `PlayRoom<VideoGame>`
```

■ Generic Implementation – Syntax

```
struct Name<T: Trait1 + Trait2, U: Trait3> {  
    field1: T,  
    field2: U,  
}
```

```
impl<T: Trait1 + Trait2, U: Trait3> Name<T, U> {  
    fn func(&self, arg1: T, arg2: U) {}  
}
```

Generic Implementation – Syntax

```
struct Name<T, U>
where
    T: Trait1 + Trait2,
    U: Trait3,
{
    field1: T,
    field2: U,
}

impl<T, U> Name<T, U>
where
    T: Trait1 + Trait2,
    U: Trait3,
{
    fn func(&self, arg1: T, arg2: U) {}
}
```

■ Generic Implementation – Example

```
trait Game {  
    fn name(&self) -> String;  
}  
  
struct PlayRoom<T: Game> {  
    game: T,  
}  
  
impl<T: Game> PlayRoom<T> {  
    pub fn game_info(&self) {  
        println!("{}", self.game.name());  
    }  
}
```


■ Generic Implementation – Usage

```
impl<T: Game> PlayRoom<T> {  
    pub fn game_info(&self) { ...  
    }  
}
```

```
let video_room = PlayRoom {  
    game: VideoGame::Xbox,  
};
```

```
let board_room = PlayRoom {  
    game: BoardGame::Monopoly,  
};
```

```
video_room.game_info();  
board_room.game_info();
```

■ Recap

- ◆ Generic structs can have both concrete and generic implementation blocks
 - Concrete implementations only apply to the type indicated in the angle braces
 - Generic implementations apply to all types that also implement the indicated trait
- ◆ Two syntaxes available

■ Recap – Syntax

```
impl<T: Trait1 + Trait2, U: Trait3> Name<T, U> {  
    fn func(&self, arg1: T, arg2: U) {}  
}
```

```
impl<T, U> Name<T, U>  
where  
    T: Trait1 + Trait2,  
    U: Trait3,  
{  
    fn func(&self, arg1: T, arg2: U) {}  
}
```