

Shared Functionality | Trait Objects

■ Trait Object Basics

- ◆ Dynamically allocated object
 - “Runtime generics”
 - ▶ More flexible than generics
 - ▶ “Dynamic Dispatch” vs “Static Dispatch”
- ◆ Allows mixed types in a collection
 - Easier to work with similar data types
 - Polymorphic program behavior
 - ▶ Dynamically change program behavior at runtime
 - ▶ Easily add new behaviors just by creating a new ***struct***
- ◆ Small performance penalty

■ Creating a Trait Object

```
trait Clicky {  
    fn click(&self);  
}
```

```
struct Keyboard;
```

```
impl Clicky for Keyboard {  
    fn click(&self) {  
        println!("click clack");  
    }  
}
```

■ Creating a Trait Object

```
let keeb = Keyboard;
```

```
let keeb_obj: &dyn Clicky = &keeb;
```

```
let keeb: &dyn Clicky = &Keyboard;
```

```
let keeb: Box<dyn Clicky> = Box::new(Keyboard);
```

■ Trait Object Parameter – Borrow

```
fn borrow_clicky(obj: &dyn Clicky) {  
    |   obj.click();  
    }  
}
```

```
let keeb = Keyboard;  
borrow_clicky(&keeb);
```

■ Trait Object Parameter – Move

```
fn move_clicky(obj: Box<dyn Clicky>) {  
    | obj.click();  
}
```

```
let keeb = Box::new(Keyboard);  
move_clicky(keeb);
```

Heterogeneous Vector

```
struct Mouse;  
impl Clicky for Mouse {  
    fn click(&self) {  
        println!("click");  
    }  
}
```

```
let keeb: Box<dyn Clicky> = Box::new(Keyboard);  
let mouse: Box<dyn Clicky> = Box::new(Mouse);  
let clickers = vec![keeb, mouse];
```

```
let keeb = Box::new(Keyboard);  
let mouse = Box::new(Mouse);  
let clickers: Vec<Box<dyn Clicky>> = vec![keeb, mouse];
```

Heterogeneous Vector

```
fn make_clicks(clickkeys: Vec<Box<dyn Clicky>>) {  
    for clicker in clickkeys {  
        clicker.click();  
    }  
}
```

```
let keeb = Box::new(Keyboard);
```

```
let mouse = Box::new(Mouse);
```

```
let clickers: Vec<Box<dyn Clicky>> = vec![keeb, mouse];
```

```
make_clicks(clickers);
```

Recap

- ◆ Trait objects allow for composite collections
- ◆ Slightly less performant than using generics
- ◆ Use the *dyn* keyword when working with trait objects
- ◆ Trait objects can be borrowed using a reference, or moved using a box
 - Usually want to use a box when storing trait objects in a Vector