Traits and Generics

Ryan Eberhardt and Armin Namavari
April 21, 2020

The Plan for Today

- Introduce traits
- Introduce generics
- See examples in real world systems! (if time permits)
- Next time: wrap up traits/generics + discuss smart pointers!
- Next week: Multiprocessing pitfalls and multiprocessing in Rust!
- *Heads up: I will be switching between slides/code — hopefully the context switching won't incur too much overhead.

Please ask Questions!

- Or else I will happily blast through the slides
- Feel free to unmute yourself
- I can also look for hands when I pause for questions

Grouping Related Functionality Together

- What are some ways you've seen this in other languages?

```
#pragma once
namespace cs20a
{
	class shape
	{
	public:
		shape();

		virtual double area() = 0;
		virtual double circumference() = 0;
		virtual std::string getDescription();
	};
}
```

```
#include <iostream>
#include "shape.h"
#include <string>
namespace cs20a
{
	shape::shape()
	{ }

	std::string shape::getDescription()
	{ return "undefined shape"; }
}
```

```
//This class implements Comparable interface
public class Player implements Comparable{
    private int ranking;
    private String name;
    private int age;
    private String country;


    public Player() {

    public Player(int ranking, String name,int age, String country) {

    //getters and setters

    @Override
    //Comparison by Ranking
    public int compareTo(Object obj) {
        Player p=(Player)obj;
            if(this.getRanking()==p.getRanking()){
                return 0;
            }else{
                return (this.getRanking()-p.getRanking());
            }
        }
```

Sources: https://www.chegg.com/homework-help/questions-and-answers/c-programming-create-required-classes-header-implementation-files-implement-following-hier-q18713018, https://qph.fs.quoracdn.net/main-qimg-4e054f260faefa31e66e02d2345091f3.webp

Traits — Some Common Ones in Rust

- What can this type do?
  - Display (lecture example)
  - Clone/Copy (exercises)
  - Iterator/IntoIterator (exercises)
  - Eq/Partial Eq (exercises)
- Allows us to override functionality
  - Drop (lecture example)
  - Deref (later)
- Allows us to define default implementations
  - ToString (will see later how this interacts with Display)
- Allows us to overload operators
  - +, -, *, /, >, <, ==, !=, etc. (lecture example)

Linked List Traits

- Playground example here (from last lectures notes)
- Let’s see Display and Drop in action!

Deriving Traits

- Provide reasonable default implementations
- Common w/ Eq/PartialEq, Copy/Clone, Debug
  - PartialEq for f64: NaN != NaN
- Point playground example

```
pub trait Copy: Clone {
    // Empty.
}
```

The following is a list of derivable traits:

- Comparison traits: `Eq`, `PartialEq`, `Ord`, `PartialOrd`.
- `Clone`, to create `T` from `&T` via a copy.
- `Copy`, to give a type 'copy semantics' instead of 'move semantics'.
- `Hash`, to compute a hash from `&T`.
- `Default`, to create an empty instance of a data type.
- `Debug`, to format a value using the `{:?}` formatter.

Defining Your Own Traits

- What if we wanted a trait to describe things that have (L2) norms? e.g. Vec<f64>, or say our new Point type.
- ComputeNorm example with Point (also, overloading “+”)
  - Playground link
  - Associated type with Add — will pop up with iterators too!

Generics

- You've seen them before: Vec<T>, Box<T>, Option<T>, Result<T, E>
- Soon: LinkedList<T> (exercises)
- MyOption<T>, MatchingPair<T>
    - Playground link

Trait Bounds and Syntax in Functions

- Sometimes we want to specify trait bounds — i.e. for what kinds of types can we call this function?
  - Generalize previous example: playground link
- identity_fn, print_excited, print_min
  - Playground link

Trait Bounds in ToString

```
impl<T: fmt::Display + ?Sized> ToString for T {
    #[inline]
    default fn to_string(&self) -> String {
        use fmt::Write;
        let mut buf = String::new();
        buf.write_fmt(format_args!("{}", self))
            .expect("a Display implementation returned an error
unexpectedly");
        buf.shrink_to_fit();
        buf
    }
}
```

Zero Cost Abstractions

- How expensive is it to keep track of all this information?
- Thanks to the magic of the Rust compiler, it's not too expensive!
- e.g. Generics => multiple versions of compiled code for different types
  - Compiler infers which one to use based on type of a piece of data
- Read more here

Examples in Real World Systems (e.g. Tock)

- Tock is an embedded OS for low-powered IoT (Internet of Things) devices
- It's written in Rust!
- You can see traits everywhere
  - Here is just one file
  - Using traits to define a syscall interface
- You can't do anything like this in C!

Additional Reading

- CS242 Notes on Traits
- About Common Rust Traits
- The Rust Book on Traits