Отчет по лабораторной работе №3 по диспиплине “Парадигмы и конструкции языков программирования”

shapes.rs

use std::fmt;

pub enum SquareResult {

AbstractMethod,

Ok(*f64*)

}

impl fmt::*Display* for SquareResult {

fn fmt(&self, f: &mut fmt::Formatter<'\_>) -> fmt::Result {

match self {

SquareResult::AbstractMethod => write!(f, "Abstract method"),

SquareResult::Ok(value) => write!(f, "{value}")

}

}

}

pub struct Figure { }

impl Figure {

pub fn new() -> Self {

Figure {}

}

}

pub trait *Repr* {

fn repr(&self) -> String;

}

pub trait *Square* {

fn get\_square(&self) -> SquareResult;

}

impl *Square* for Figure {

fn get\_square(&self) -> SquareResult {

SquareResult::AbstractMethod

}

}

impl *Repr* for Figure {

fn repr(&self) -> String {

String::from("------------\n[Figure] abstract struct\n------------")

}

}

shapes/square.rs

use std::borrow::*Borrow*;

use super::shape::{Figure, *Square*, SquareResult, *Repr*};

use crate::color::Color;

pub struct Rectangle {

**figure**: Figure,

**color**: Color,

**length**: *f64*,

**width**: *f64*,

}

impl Rectangle {

pub fn new(color: Color, length: *f64*, width: *f64*) -> Self {

Rectangle { **figure**: Figure::new(), **color**: color, **length**: length, **width**: width }

}

pub fn get\_length(&self) -> *f64* {

self.**length**

}

pub fn get\_width(&self) -> *f64* {

self.**width**

}

pub fn get\_color(&self) -> String {

self.**color**.get\_color()

}

}

impl *Square* for Rectangle {

fn get\_square(&self) -> SquareResult {

let square: *f64* = self.**length** \* self.**width**;

SquareResult::Ok(square)

}

}

impl *Repr* for Rectangle {

fn repr(&self) -> String {

format!("------------\n[Class]: Rectangle\n[Color]: {}\n[Length]: {}\n[Width]: {}\n[Square]: {}\n------------",

self.**color**.get\_color(), self.**length**, self.**width**, self.get\_square())

}

}

shapes/rectangle.rs

use std::borrow::*Borrow*;

use super::shape::{Figure, *Square*, SquareResult, *Repr*};

use crate::color::Color;

pub struct Rectangle {

**figure**: Figure,

**color**: Color,

**length**: *f64*,

**width**: *f64*,

}

impl Rectangle {

pub fn new(color: Color, length: *f64*, width: *f64*) -> Self {

Rectangle { **figure**: Figure::new(), **color**: color, **length**: length, **width**: width }

}

pub fn get\_length(&self) -> *f64* {

self.**length**

}

pub fn get\_width(&self) -> *f64* {

self.**width**

}

pub fn get\_color(&self) -> String {

self.**color**.get\_color()

}

}

impl *Square* for Rectangle {

fn get\_square(&self) -> SquareResult {

let square: *f64* = self.**length** \* self.**width**;

SquareResult::Ok(square)

}

}

impl *Repr* for Rectangle {

fn repr(&self) -> String {

format!("------------\n[Class]: Rectangle\n[Color]: {}\n[Length]: {}\n[Width]: {}\n[Square]: {}\n------------",

self.**color**.get\_color(), self.**length**, self.**width**, self.get\_square())

}

}

shapes/circle.rs

use std::f64::consts::PI;

use super::shape::{Figure, *Square*, *Repr*, SquareResult};

use crate::color::Color;

pub struct Circle {

**figure**: Figure,

**radius**: *f64*,

**color**: Color

}

impl Circle {

pub fn new(color: Color, radius: *f64*) -> Self {

Circle { **figure**: Figure::new(), **radius**: radius, **color**: color }

}

}

impl *Square* for Circle {

fn get\_square(&self) -> SquareResult {

let square: *f64* = PI \* self.**radius** \* self.**radius**;

SquareResult::Ok(square)

}

}

impl *Repr* for Circle {

fn repr(&self) -> String {

format!("------------\n[Class]: Circle\n[Color]: {}\n[Raduis]: {}\n[Square]: {}\n------------",

self.**color**.get\_color(), self.**radius**, self.get\_square())

}

}

color.rs

pub struct Color {

**r**: *u8*,

**g**: *u8*,

**b**: *u8*,

}

impl Color {

pub fn new(r: *u8*, g: *u8*, b: *u8*) -> Self {

Color { **r**: r, **g**: g, **b**: b }

}

pub fn get\_color(&self) -> String {

format!("rgb({} {} {})", self.**r**, self.**g**, self.**b**)

}

}

tests.rs

#[cfg(test)]

mod tests {

use crate::shapes::{circle::Circle, rectangle::Rectangle, shape::*Repr*, square::SquareFig};

use crate::color::Color;

#[test]

fn rect() {

let N: *f64* = 16.0; *// N = 16 по номеру варианта*

let rect: Rectangle = Rectangle::new(Color::new(0, 0, 255), N, N);

assert\_eq!(rect.repr(), String::from("------------\n[Class]: Rectangle\n[Color]: rgb(0 0 255)\n[Length]: 16\n[Width]: 16\n[Square]: 256\n------------"))

}

#[test]

fn circle() {

let N: *f64* = 16.0; *// N = 16 по номеру варианта*

let circle: Circle = Circle::new(Color::new(0, 255, 0), N);

assert\_eq!(circle.repr(), String::from("------------\n[Class]: Circle\n[Color]: rgb(0 255 0)\n[Raduis]: 16\n[Square]: 804.247719318987\n------------"))

}

#[test]

fn square() {

let N: *f64* = 16.0; *// N = 16 по номеру варианта*

let square: SquareFig = SquareFig::new(Color::new(255, 0, 0), N);

assert\_eq!(square.repr(), String::from("------------\n[Class]: Square\n[Color]: rgb(255 0 0)\n[Length]: 16\n[Square]: 256\n------------"))

}

}

main.rs

use rand::*Rng*;

pub mod shapes;

pub mod color;

pub mod tests;

use color::Color;

use shapes::{circle::Circle, rectangle::Rectangle, shape::*Repr*, square::SquareFig};

fn main() {

let mut rng = rand::thread\_rng();

*// let N: f64 = rng.gen\_range(5.0..50.0); // N = 16 по номеру варианта*

let N: *f64* = 16.0;

let rect: Rectangle = Rectangle::new(Color::new(0, 0, 255), N, N);

let circle: Circle = Circle::new(Color::new(0, 255, 0), N);

let square: SquareFig = SquareFig::new(Color::new(255, 0, 0), N);

println!("{}", rect.repr());

println!("{}", circle.repr());

println!("{}", square.repr());

}