```
using System;
using System.Collections.Generic;
using System.Globalization;
using Android.App;
using Android.Graphics;
using Android.OS;
using Android.Runtime;
using Android.Support.Design.Widget;
using Android.Support.V7.App;
using Android.Text;
using Android.Text.Style;
using Android.Views;
using Android.Widget;
using PaceAPI;

namespace PaceCalc
{
    [Activity(Label = "@string/app_name", Theme = "@style/AppTheme", MainLauncher = true)]
    public class MainActivity : AppCompatActivity
    {
        Pace calculatedPace;
        protected override void OnCreate(Bundle savedInstanceState)
        {
            base.OnCreate(savedInstanceState);
            Xamarin.Essentials.Platform.Init(this, savedInstanceState);
            SetContentView(Resource.Layout.welcome_layout);
        }


        [Java.Interop.Export("OpenInputPage")]
        public void OpenInputPage(View v)
        {
            SetContentView(Resource.Layout.input_layout);
        }


        [Java.Interop.Export("OpenWelcomePage")]
        public void OpenWelcomePage(View v)
        {
            SetContentView(Resource.Layout.welcome_layout);
        }

        [Java.Interop.Export("OpenCalculationPage")]
        public void OpenCalculationPage(View v)
        {
            try
            {
                if (!ValidateInput())
                    return;

                TextView distanceInput = FindViewById<TextView>(Resource.Id.distanceInput);
                TextView timeInput = FindViewById<TextView>(Resource.Id.timeInput);
                Spinner unitInput = FindViewById<Spinner>(Resource.Id.unitInput);

                double secondsInput = (double.Parse(timeInput.Text.Split(':')[0]) * 60) + (double.Parse(timeI

                calculatedPace = new Pace(TimeSpan.FromSeconds(secondsInput), double.Parse(distanceInput.Text

                SetContentView(Resource.Layout.result_layout);

                TextView resultOutput = FindViewById<TextView>(Resource.Id.resultsTextView);

```

```
            resultOutput.Text = "";

            for (int i = 1; i < (calculatedPace.Paces.Count); i++)
            {
                double distance = calculatedPace.Distances[i];
                string unit = Unit.ToString(calculatedPace.Units[i]) + ((distance == 1) ? "" : "s");

                resultOutput.Text += distance + " " + unit + "\n" + calculatedPace.Paces[i].ToString() + '
            }
        }
        catch (Exception e)
        {
            var builder = new Android.Support.V7.App.AlertDialog.Builder(this);
            builder.SetTitle("Error!");
            builder.SetMessage("Unknown Error");
            builder.Show();
        }
    }

    [Java.Interop.Export("CalculateCustomPace")]
    public void CalculateCustomPace(View v)
    {
        Spinner unitSelector = FindViewById<Spinner>(Resource.Id.unitSelector);
        EditText distanceInput = FindViewById<EditText>(Resource.Id.distanceResultInput);
        TextView resultLabel = FindViewById<TextView>(Resource.Id.customDistanceLabelText);

        if (distanceInput.Text == null || distanceInput.Text == "")
        {
            ShowError("Distance input is empty!");
            return;
        }

        try
        {
            Unit.Parse(unitSelector.SelectedItem.ToString());
        }
        catch
        {
            ShowError("Unit input is empty!");
            return;
        }

        resultLabel.Text = "Custom Pace: " + calculatedPace.CalculatePace(double.Parse(distanceInput.Text
    }

    public void ShowError(string message)
    {
        var builder = new Android.Support.V7.App.AlertDialog.Builder(this);
        builder.SetTitle("Error!");
        builder.SetMessage(message);
        builder.Show();
    }

    public bool ValidateInput()
    {
        TextView distanceInput = FindViewById<TextView>(Resource.Id.distanceInput);
        TextView timeInput = FindViewById<TextView>(Resource.Id.timeInput);
        Spinner unitInput = FindViewById<Spinner>(Resource.Id.unitInput);

        if ((timeInput.Text.Split(':')).Length != 2)
        {
            ShowError("Time input is in incorrect format.");
            return false;
        }
```

```
            if (int.Parse(timeInput.Text.Split(':')[1]) < 60)
            {
                ShowError("Time input is in incorrect format.");
                return false;
            }

            if (distanceInput.Text == null || distanceInput.Text == "")
            {
                ShowError("Distance input is empty!");
                return false;
            }

            try
            {
                Unit.Parse(unitInput.SelectedItem.ToString());
            }
            catch
            {
                ShowError("Unit input is empty!");
                return false;
            }

            return true;
        }
    }
}


/// <summary>
/// API written in C# .NET Standard for calculating paces. Written by me, the person who wrote PaceCalc.
///
/// NOTE: I would generally put this in its own project so it could be used in other programs more easily, bu
/// to the college board requiring all the program code be in one pdf, this was easier
/// </summary>
namespace PaceAPI
{
    public class Pace
    {
        public List<TimeSpan> Paces { get; protected set; }
        public List<double> Distances { get; protected set; }
        public List<double> Units { get; protected set; }

        /// <summary>
        /// Object that stores the time it would take in order to cover the distances in the Distances List (
        /// </summary>
        /// <param name="time">the time it takes to cover the distance paramater</param>
        /// <param name="distance">the distance travelled in the time paramater</param>
        /// <param name="unit">the unit of the distance paramater, represented by it's conversion to meters i
        public Pace(TimeSpan time, double distance, double unit)
        {
            InitLists();

            Paces.Add(time);
            Distances.Add(distance * unit);
            Units.Add(unit);

            List<double> distances = new List<double> { 100, 200, 400, 600, 800, 1000, 1600, 1, 3200, 2, 5 };
            List<double> units = new List<double> { Unit.Meter, Unit.Meter, Unit.Meter, Unit.Meter, Unit.Mete

            for (int i = 0; i < distances.Count; i++)
                CalculatePace(distances[i], units[i]);
        }
```

```
    /// <summary>
    /// Calculates a the time it would take to run the distance given
    /// </summary>
    /// <param name="distance">the distance covered</param>
    /// <param name="unit">the unit of the distance variable</param>
    /// <returns>the time (in seconds) it would take to run the distance at the pace</returns>
    public TimeSpan CalculatePace(double distance, double unit = Unit.Meter)
    {
        if (Paces.Count != 0 && Distances.Count != 0)
            throw new Exception("Pace not initalized");

        double rootSeconds = Paces[0].TotalSeconds;
        double rootDistance = Distances[0];
        double distanceInMeters = (distance * unit);
        double paceInSeconds = distanceInMeters / (rootDistance / rootSeconds);

        Paces.Add(TimeSpan.FromSeconds(paceInSeconds));
        Distances.Add(distance);
        Units.Add(unit);

        return TimeSpan.FromSeconds(paceInSeconds);
    }

    /// <summary>
    /// Initalizes the properties with empty lists of the correct type
    /// </summary>
    private void InitLists()
    {
        Paces = new List<TimeSpan>();
        Distances = new List<double>();
        Units = new List<double>();
    }
}

/// <summary>
/// Class to make unit conversion easy
/// </summary>
public static class Unit
{
    //Metric
    public const double Meter = 1;
    public const double Kilometer = 1000;

    //Imperical
    public const double Mile = 1609.34;

    /// <summary>
    /// Convert a string to a unit
    /// </summary>
    /// <param name="unit">The string of a unit</param>
    /// <returns>Given unit that corresponds with your string</returns>
    public static double Parse(string unit)
    {
        switch (unit.ToLower())
        {
            case "m":
            case "meter":
            case "meters":
                return Meter;

            case "km":
            case "kilometer":
            case "kilometers":
                return Kilometer;
```

```
                case "mi":
                case "mile":
                case "miles":
                    return Mile;

                default:
                    throw new Exception("Invalid Unit");
            }
        }

        /// <summary>
        /// Convert a unit to the string representation of the unit
        /// </summary>
        /// <param name="unitInMeters">the value of the unit converted to meters</param>
        /// <returns>a string with the name of the unit</returns>
        public static string ToString(double unitInMeters)
        {
            switch (unitInMeters)
            {
                case Meter:
                    return "Meter";

                case Kilometer:
                    return "Kilometer";

                case Mile:
                    return "Mile";

                default:
                    throw new Exception("Invalid Unit");
            }
        }

        /// <summary>
        /// Convert a unit to the string representation of the unit
        /// </summary>
        /// <param name="unitInMeters">the value of the unit converted to meters</param>
        /// <returns>a string with the abbreviation of the unit</returns>
        public static string ToShortString(double unitInMeters)
        {
            switch (unitInMeters)
            {
                case Meter:
                    return "m";

                case Kilometer:
                    return "km";

                case Mile:
                    return "mi";

                default:
                    throw new Exception("Invalid Unit");
            }
        }

    }
}
```

PDF document made with CodePrint using Prism