1. Creating from Scratch:

create a string util package in perl

Observation: copilot automatically created the function with basic logic.

package StringUtil;

sub new {

  my $class = shift;

  my $self = {};

  bless $self, $class;

  return $self;

}

sub reverse\_string {

  my ($self, $string) = @\_;

  return scalar reverse $string;

}

sub uppercase\_string {

  my ($self, $string) = @\_;

  return uc $string;

}

sub lowercase\_string {

  my ($self, $string) = @\_;

  return lc $string;

}

1;

1. Fine-tuning the prompt:  
   add more cases to the string manipulator
2. sub remove\_whitespace {
3. my ($self, $string) = @\_;
4. $string =~ s/\s+//g;
5. return $string;
6. }
7. sub count\_words {
8. my ($self, $string) = @\_;
9. my @words = split /\s+/, $string;
10. return scalar @words;
11. }
12. sub replace\_substring {
13. my ($self, $string, $old, $new) = @\_;
14. $string =~ s/\Q$old\E/$new/g;
15. return $string;
16. }
17. 1;

3. We can even ask the co-pilot to convert the code to use different modules.

convert the perl DBUtil based on oracle usage

use DBI;

sub new {

  my ($class, $dsn, $user, $password) = @\_;

  my $self = {

    dsn => $dsn,

    user => $user,

    password => $password,

    dbh => undef

  };

  bless $self, $class;

  return $self;

}

sub get\_table\_names {

  my ($self) = @\_;

  my $sth = $self->{dbh}->prepare("SHOW TABLES")

    or die "Could not prepare query: " . $self->{dbh}->errstr;

  $sth->execute()

    or die "Could not execute query: " . $sth->errstr;

  my @table\_names;

  while (my $row = $sth->fetchrow\_arrayref()) {

    push @table\_names, $row->[0];

  }

  return \@table\_names;

}

use DBD::Oracle;

sub new {

  my ($class, $dsn, $user, $password) = @\_;

  my $self = {

    dsn => $dsn,

    user => $user,

    password => $password,

    dbh => undef

  };

  bless $self, $class;

  return $self;

}

sub get\_table\_names {

  my ($self) = @\_;

  my $sth = $self->{dbh}->prepare("SELECT table\_name FROM all\_tables")

    or die "Could not prepare query: " . $self->{dbh}->errstr;

  $sth->execute()

    or die "Could not execute query: " . $sth->errstr;

  my @table\_names;

  while (my $row = $sth->fetchrow\_arrayref()) {

    push @table\_names, $row->[0];

  }

  return \@table\_names;

}

4. Teach once to the co-pilot, then it will automatically follow for other code.

add validation to all function to check if we got required input

package DBUtil;

use DBI;

sub new {

  my ($class, $dsn, $user, $password) = @\_;

  die "Missing required input: dsn" unless defined $dsn;

  die "Missing required input: user" unless defined $user;

  die "Missing required input: password" unless defined $password;

  my $self = {

    dsn => $dsn,

    user => $user,

    password => $password,

    dbh => undef

  };

  bless $self, $class;

  return $self;

}

add update and delete function to the dbutil module

sub update\_row {

  my ($self, $table\_name, $data, $where) = @\_;

  die "No database handle exists" unless defined $self->{dbh};

  die "Missing required input: table\_name" unless defined $table\_name;

  die "Missing required input: data" unless defined $data;

  die "Missing required input: where" unless defined $where;

  my $set\_clause = join(", ", map { "$\_ = :$\_" } keys %$data);

  my $where\_clause = join(" AND ", map { "$\_ = :$\_" } keys %$where);

  my $sth = $self->{dbh}->prepare("UPDATE $table\_name SET $set\_clause WHERE $where\_clause")

    or die "Could not prepare query: " . $self->{dbh}->errstr;

  $sth->execute(%$data, %$where)

    or die "Could not execute query: " . $sth->errstr;

}

5. HIPAA standard can be implemented within copilot

write procedure code validation based on HIPAA standard with oops concepts

package DiagnosisCode;

use strict;

use warnings;

use Carp qw(croak);

sub new {

    my ($class, $code) = @\_;

    croak "Missing required input: code" unless defined $code;

    my $self = {

        code => $code

    };

    bless $self, $class;

    return $self;

}

sub validate {

    my ($self) = @\_;

    my $code = $self->{code};

    # Check if the code is alphanumeric and has a length of 3

    unless ($code =~ /^[A-Z0-9]{3}$/) {

        croak "Invalid diagnosis code: $code";

    }

    # Check if the first character is a letter

    unless ($code =~ /^[A-Z]/) {

        croak "Invalid diagnosis code: $code";

    }

    # Check if the second and third characters are digits between 0 and 9

    unless ($code =~ /^[A-Z][0-9]{2}$/) {

        croak "Invalid diagnosis code: $code";

    }

    return 1;

}

1;

6. Different solutions each time:

improve the code

package PatientUtil;

use strict;

use warnings;

use Carp qw(croak);

sub new {

    my ($class, %params) = @\_;

    my @required\_params = qw(first\_name last\_name dob gender address phone\_number);

    foreach my $param (@required\_params) {

        croak "Missing required input: $param" unless defined $params{$param};

    }

    my $self = \%params;

    bless $self, $class;

    return $self;

}

7. Converting the Perl to java

convert the code to java

public class DiagnosisCode {

    private static Map<String, String> diagnosisCodes = new HashMap<String, String>() {{

        put("A00", "Cholera");

        put("A01", "Typhoid and paratyphoid fevers");

        put("A02", "Other salmonella infections");

        // ...

    }};

    public static void validate(String code) throws Exception {

        if (!code.matches("^[A-Z0-9]{3}$")) {

            throw new Exception("Invalid diagnosis code: " + code);

        }

        if (!Character.isLetter(code.charAt(0))) {

            throw new Exception("Invalid diagnosis code: " + code);

        }

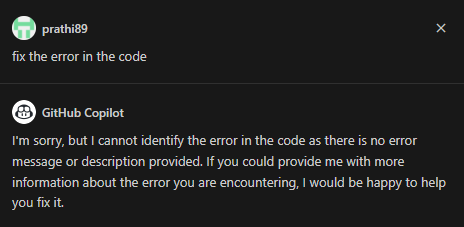
        if (!code.substring(1).matches("^[0-9]{2}$")) {

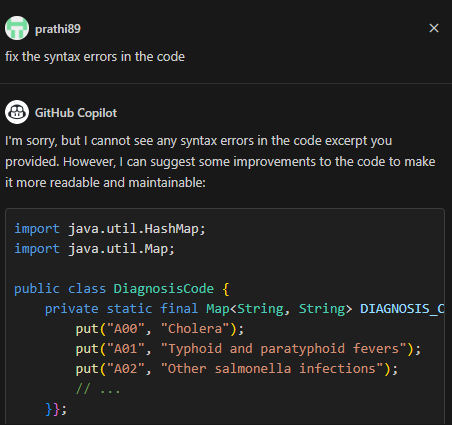
            throw new Exception("Invalid diagnosis code: " + code);

        }

    }

8. Instruct with proper prompt to fix the error.



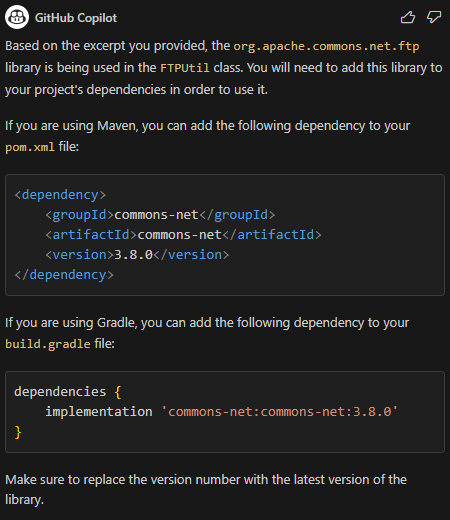




Here the code is not converted fully only part of the code was converted.

But second time with same prompt converted the entire file.

Fixing the missing library:

add the missing java library  
  


|  |  |  |
| --- | --- | --- |
| **Cases** | **Pros** | **Cons** |
| It can start from scratch | Copilot will be able to help you code from scratch with default use cases. | Since it is created with default cases, you need to be more specific on the prompt; otherwise, there will be a rework. |
| Teach once to the copilot | Copilot is designed to adapt based on the developer; once you give the proper prompt to the project, it will automatically apply the same case to the next prompt (example: error handling). | Each block can have a different use case, so sometimes it might increase the review time. |
| UserDefined modules | Even though it picks the best module, if required, we can instruct the copilot to use user-defined or desired modules. |  |
| Different solutions each time | Copilot will suggest different solution each time even if you have better approach. |  |
| Code convert | It convert all the function and variable and make sure it create equivalent logic is implemented | Sometime it miss to import required module unless we explicitly mention in the prompt. |
| Instruct with proper prompt to fix the error. |  | Even we have error sometime copilot says it cant able to identify the error unless the prompt is much clear |