Exercise 1

Topics: Unit testing, equivalence partitioning, boundary value analysis

Setup

- Connect linux-desktop.tuni.fi, note that you need to apply for the remote desktop permissions beforehand. See instructions from Moodle under exercise -section.
- Download and extract the Ex1.zip project.
- Open the project in QtCreator (under Applications -> Programming -> QtCreator 5.14.2 for TIE courses)
- Note that the Class implementation code uses some Finnish words, comments are both in English and in Finnish
    - Paivays = Date
    - Paiva = Day
    - Kuukausi = Month
    - Vuosi = Year
    - Viikonpaiva = Day of the week
- The content of the .hh-file is also available at the end of this document, with English and Finnish comments.

Planning tasks

1. Consider the parameters in methods asetaPaiva *(setDay)*, asetaKK *(setMonth)*and asetaVuosi *(setYear)*. Each of the method takes one parameter, either the new day, month or year, depending which function you are calling. From the point of view of the application field (NOT implementation, consider with black box view for this task), list the equivalent classes for the input values of each parameter.
2. Using boundary value analysis, list the testable boundary values for the same parameters.
3. Consider now the constructor of the Paivays -class. It has 3 parameters, day, month and year. Sometimes problems manifest only when the combination of the values is erroneous. As the amount of possible combinations grows fast, pairwise testing can be used to reduce the combinations. So, in this example, instead of testing all possible combinations of the 3 parameters, tests are written to cover each pair at least once. Create the pairwise combinations for testing the constructor.

   You can use https://pairwise.teremokgames.com/4s8/ to check your combinations.
4. You can use equivalent classes for internal details of the methods too. Consider etene *(advance)*-method and the parameter n. What kind of value ranges cause the control structure to choose different paths Remember to consider the combinations of parameter and the member variables.

5. Sometimes it can be useful to use the methods for return values instead of parameters. After identifying the test cases from return values, one must find out the suitable input values to test that the output is produced properly. Consider annaViikonpaiva *(getDayOfTheWeek)*-method and generate tests from the return values' point of view.

Code tasks

NOTE: Due to limited time, we will not attempt to write all possible unit tests for each task.

TIPS for writing unit test:

- Name your tests so that it is clear such as: What method is being tested, what is the scenario being tested and what is the expected behaviour.
- Keep tests simple, more complicated your code is, more probable it is to introduce bug in your test code.
- You can use Arrange, Act, Assert -pattern when writing unit tests. Arrange part: whatever you need to do before you can do the call we are testing. Act: call the method we are testing. Assert: Did the expected happen or not.

1. In Planning task 1 you found out the equivalent classes for each parameter in the methods asetaPaiva *(setDay)*, asetaKK *(setMonth)*, asetaVuosi *(setYear)*. Write 5 unit tests to cover 5 of those equivalent classes.
    - In "Arrange phase", you can use constructor to create Paivays -object with neutral date. Then you can manipulate the date with the asetaPaiva/asetaKK/asetaVuosi -methods. Project code has couple of examples for simple comparison macros available.
2. In Planning task 2 you found the boundary values for the same methods. Write 5 unit tests to cover 5 of those boundary values.
3. Add unit tests for your test group by writing implementations for handful of tests cases identified in tasks 3-5.

```
#ifndef PAIVAYS_HH
#define PAIVAYS_HH

class HuonoPaivays
{
    // Bad Date Exception
    // Tyhjä toteutus poikkeusluokalle, jotta kääntyisi
    // Empty implementation for the exception class, for compiling
};

/*!
  * \brief Yksinkertainen päiväysluokka
  * Päiväysluokka Gregoriaanisen kalenterin päiväyksille Suomessa. Vuosiluvut
ennen vuotta 1754 ovat
  * laittomia. Päivien ja kuukausien numerointi lähtee 1:stä.
  * Simple date class for Gregorian calendar in Finland. Years before 1754 are
not allowed. Date and
  * month numbering starts from 1.
  **/

// Date
class Paivays
{
public:
    /*!
     * \brief Viikonpaiva on luettelotyyppi viikonpäiville
     *
     * Viikonpaiva is enum for date names. Monday, Tuesday, Wednesday, Thursday,
Friday, Saturday, Sunday.
     */
    enum Viikonpaiva { MAANANTAI, TIISTAI, KESKIVIIKKO, TORSTAI,
PERJANTAI, LAUANTAI, SUNNUNTAI };

    /*!
   * \brief Rakentaja alustaa päiväyksen annettuun päivämäärään. Constructor for
setting specific date
   * \param p Päivä, Day
   * \param k Kuukausi, Month
   * \param v Vuosi, Year
   * \exception HuonoPaivays   Laiton vuosi, kuukausi tai päivä, tai päivä on
liian suuri kuukauteen. Illegal year, month
   * or day, or day is not within month.
   **/
     Paivays(unsigned int p, unsigned int k, unsigned int v); //Date

    /*!
   * \brief Purkaja purkaa päiväyksen. Destructor for date
   * \pre -
   **/
    ~Paivays();

    /*!
   * \brief asetaPaiva asettaa päivän. Sets day
   * \param paiva Uusi päivä. New day.
   * \pre -
   * \post Päiväyksen päivä on asetettu annetuksi (kuukausi ja vuosi ennallaan).
Day for the Paivays (date) has been set.
```

```
   *    Month and year unchanged.
   * \exception HuonoPaivays   Päivä on laiton tai liian suuri kuukauteen. Day is
illegal or not within the month.
   */

    void asetaPaiva(unsigned int paiva); // setDay

    /*!
   * \brief asetaKk asettaa kuukauden. Sets month
   * \param kuukausi Uusi kuukausi. New month.
   * \pre -
   * \post Päiväyksen kuukausi on asetettu annetuksi (päivä ja vuosi ennallaan).
Month of the Paivays (date) has been set.
   *    Day and year unchanged.
   * \exception HuonoPaivays   Kuukausi on laiton tai päivä on liian suuri
kuukauteen. Month is illegal or day is
   * not within the month.
   */

    void asetaKk(unsigned int kuukausi); // setMonth

    /*!
   * \brief asetaVuosi asettaa vuoden. Sets year
   * \param vuosi Uusi vuosi. New year.
   * \pre -
   * \post Päiväyksen vuosi on asetettu annetuksi (päivä ja kuukausi ennallaan).
Year of the Paivays (date) has been set.
   *    Day and month unchanged.
   * \exception HuonoPaivays   Vuosi on laiton tai päivä on liian suuri
kuukauteen. Year is illegal or day is not within the month.
   */

    void asetaVuosi(unsigned int vuosi); // setYear

    /*!
   * \brief annaPaiva palauttaa päivän. Returns day.
   * \return Päivämäärän päivä. Day of the date
   * \pre -
   */

    unsigned int annaPaiva() const; // getDay

    /*!
   * \brief annaKk palauttaa kuukauden. Returns month.
   * \return Päivämäärän kuukausi. Month of the date
   * \pre -
   */

    unsigned int annaKk() const; // getMonth

    /*!
   * \brief annaVuosi palauttaa vuoden. Returns year.
   * \return Päivämäärän vuosi. Year of the date.
   * \pre -
   */
    unsigned int annaVuosi() const; // getYear
```

```
    /*!
   * \brief annaViikonpaiva palauttaa päiväyksen viikonpäivän. Give day of the
week for the date.
   * \return Päivämäärän viikonpäivä. Date's day of the week.
   * \pre -
   */

    Viikonpaiva annaViikonpaiva() const; // getDayOfTheWeek


    /*!
   * \brief etene siirtää päiväystä eteenpäin. etene (advance) moves date
forward.
   * \param n montako päivää siirrytään. How many days will be moved
   * \pre -
   * \post Päiväystä on siirretty n päivää eteenpäin. Date has moved n days
forward.
   */
    void etene(unsigned int n); // advance


    /*!
   * \brief paljonkoEdella kertoo, montako päivää annettu päiväys on tätä
päiväystä edellä. How much forward. Tells how many
   * days the given date is forward compared to the date it is called for.
   * \param p Päiväys, jota vertaillaan. Date to compare
   * \return Päiväysten ero päivinä (positiivinen = p myöhemmin, negatiivinen = p
aiemmin). Difference in days. Positive value:
   *    p is later. Negative value: p is earlier.
   * \pre -
   */
    int paljonkoEdella(Paivays const& p) const; // howMuchAhead


    /*!
   * \brief onkoKarkausvuosi kertoo, onko annettu vuosi karkausvuosi. Is leap
year tells is the given year leap year
   * \param vuosi on se vuosi, josta kysytään. Year to be checked
   * \return onko karkausvuosi. Is the year leap year
   * \exception HuonoPaivays   Vuosi on laiton. Illegal year
   */
    static bool onkoKarkausvuosi(unsigned int vuosi); // isLeapYear


    /*!
   * \brief kuukaudenPituus kertoo annetun vuoden annetun kuukauden pituuden.
Length of the month tells the month's
   * length in the given year
   * \param kuukausi on se kuukausi, josta kysytään. Month to be checked
   * \param vuosi on se vuosi, josta kysytään. Year to be checked.
   * \return kuukauden pituus. Length of the month.
   * \exception HuonoPaivays   Vuosi tai kuukausi on laiton. Illegal year or
month.
   */
    static unsigned int kuukaudenPituus(unsigned int kuukausi, unsigned
int vuosi); // lengthOfMonth
```

```
    /*!
   * \brief vuodenPituus kertoo annetun vuoden pituuden. Length of the year tells
the year's length.
   * \param vuosi on se vuosi, josta kysytään. Year to be checked
   * \return vuoden pituus. Length of the year
   * \exception HuonoPaivays   Vuosi on laiton. Illegal year.
   */
    static unsigned int vuodenPituus(unsigned int vuosi); //lengthOfYear


private:
    unsigned int paiva_;    //!< Päivämäärän päivä Date's day
    unsigned int kuukausi_; //!< Päivämäärän kuukausi Date's month
    unsigned int vuosi_;    //!< Päivämäärän vuosi Date's year

    static unsigned int const KUUKAUDET[12]; //!< Kuukausien
pituustaulukko. Array for month lengths.
};

#endif
```