Discussion-2 Variables and Data Types

Discussion Topic:

Suppose you are building a Java application for storing the names and ages of children in a family. Describe the strategy that you would use in order to determine the data types you would need for your application. Provide three (3) data elements and the corresponding data types that you would use in your application. Provide a rationale for your response and provide constructive feedback on strategies and rationales posted by your peers.

My Post:

Hello class,

In a Java application, when considering data types needed for storing the names and ages of children in a family, I would carefully choose the variable data types based on the nature of the data that need to be stored.

- First, I would start by defining what information needs to be stored. This includes details like names, ages, and possibly other demographic (family income) or identification data (race).
- Secondly, I would assess the nature of each data element, including its format (numeric, text, date).
- Thirdly, based on the nature and requirements of each data element, I would select data types that meet the necessary storage capacity, performance, and functionality requirements (e.g., for numeric types, choose from short, int, long, float, or double).

The steps above, in my opinion, ensure that each piece of data is stored in an appropriate format, maintaining and/or improving the reliability and performance of an application.

Below are my data type selections needed for storing the names and ages of children in a family:

1. First Name (String):
    - Characteristics: Text (textual) includes letters, it may also include spaces, hyphens, and apostrophes.
    - Data Type Choice: String
    - Rationale: The String type is flexible for handling text of variable length and different characters, making it ideal for storing names.
      "Note: The String class is immutable, so that once it is created a String object cannot be changed. The String class has a number of methods. Since strings are immutable, what these methods really do is create and return a new string that contains the result of the operation."(Oracle (a), n.d., Strings)
2. Last Name (String):
    - Characteristics: Similar to the first name.
    - Data Type Choice: String
    - Rationale: Same as first name, it also allows sorting and searching by last name, if needed.
3. Age (short):

- Characteristics: Whole number, usually ranging from 0 to 18 for children, if the children are minors.
- Data Type Choice: short
- Rationale: The int type supports the range of typical child ages or the usual span of human life, also ages are commonly described using whole numbers. This type provides adequate capacity for this application without using unnecessary memory.
  "The short data type is a 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive). As with byte, the same guidelines apply: you can use a short to save memory in large arrays, in situations where the memory savings actually matters." (Oracle (b), n.d., Primitive Data Types)

Additional Data Elements to Consider

4. Date of Birth (LocalDate):
   - Characteristics: date information without time.
   - Data Type Choice: LocalDate
   - Rationale: LocalDate provides methods for date arithmetic and age calculation. Oracle LocalDate Documentation.
5. Gender (enum):
   - Characteristics: Categorical data.
   - Data Type Choice: enum for Gender (e.g., MALE, FEMALE, OTHER)
   - Rationale: Using an enum ensures data integrity by restricting gender to predefined values, which is safer and cleaner than using strings. Oracle Enum Documentation.

Finally, I will use modular programming principles, and I would create a family class to store the names and ages of the children.
Here is a code example of it:

Family.java

```
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

public class Family {
    private static int nextFamilyId = 1;  // Static variable for incrementing family ID
    private final int familyId;
    private List<Child> children;

    // Constructor for Family class
    public Family() {
        this.familyId = nextFamilyId++;
        this.children = new ArrayList<>();
    }

    // adds a child to the family
    public void addChild(String firstName, String lastName, short age, LocalDate dob, Gender gender) {
        children.add(new Child(firstName, lastName, age, dob, gender));
    }

    // Getters
    public int getFamilyId() {
        return familyId;
    }
```

```
    public List<Child> getChildren() {
        return new ArrayList<>(children);  // Return a copy of the children list
    }

    // Nested Child class
    private static class Child {
        private String firstName;
        private String lastName;
        private short age;
        private LocalDate dateOfBirth;
        private Gender gender;

        // Constructor for Child class
        public Child(String firstName, String lastName, short age, LocalDate dob, Gender gender) {
            this.firstName = firstName;
            this.lastName = lastName;
            this.age = age;
            this.dateOfBirth = dob;
            this.gender = gender;
        }

        // Getters
        public String getFirstName() {
            return firstName;
        }

        public String getLastName() {
            return lastName;
        }

        public short getAge() {
            return age;
        }

        public LocalDate getDateOfBirth() {
            return dateOfBirth;
        }

        public Gender getGender() {
            return gender;
        }
    }

    // Gender enumeration
    public enum Gender {
        MALE, FEMALE, OTHER
    }
}
```

Main.java

```
import java.time.LocalDate;

public class Main {

         public static void main(String[] args) {
                 Family family = new Family();
        family.addChild("John", "Doe", (short)10, LocalDate.of(2013, 3, 15), Family.Gender.MALE);
        family.addChild("Jane", "Doe", (short)8, LocalDate.of(2015, 6, 21), Family.Gender.FEMALE);

        System.out.println("Family ID: " + family.getFamilyId());
        for (Family.Child child : family.getChildren()) {
            System.out.println("Child: " + child.getFirstName() + " " + child.getLastName() +
                               ", Age: " + child.getAge() +
```

```
                                ", DOB: " + child.getDateOfBirth() +
                                ", Gender: " + child.getGender());
                }

        }
}
```

Console Output:

```
Family ID: 1
Child: John Doe, Age: 10, DOB: 2013-03-15, Gender: MALE
Child: Jane Doe, Age: 8, DOB: 2015-06-21, Gender: FEMALE
```

-Alex

References:

Oracle (a), (n.d.). The Java™ Tutorials - Strings. Oracle Java Documentation. https://docs.oracle.com/javase/tutorial/java/data/strings.html

Oracle (b), (n.d.). The Java™ Tutorials - Primitive Data Types. Oracle Java Documentation. https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html