박성준

14장 점진적 개선

Arg 클래스

작동 방식 : 사전에 주어진 Schema로 argument를 식별 & 저장

Arg(String Schema, String[] args)

<Schema>

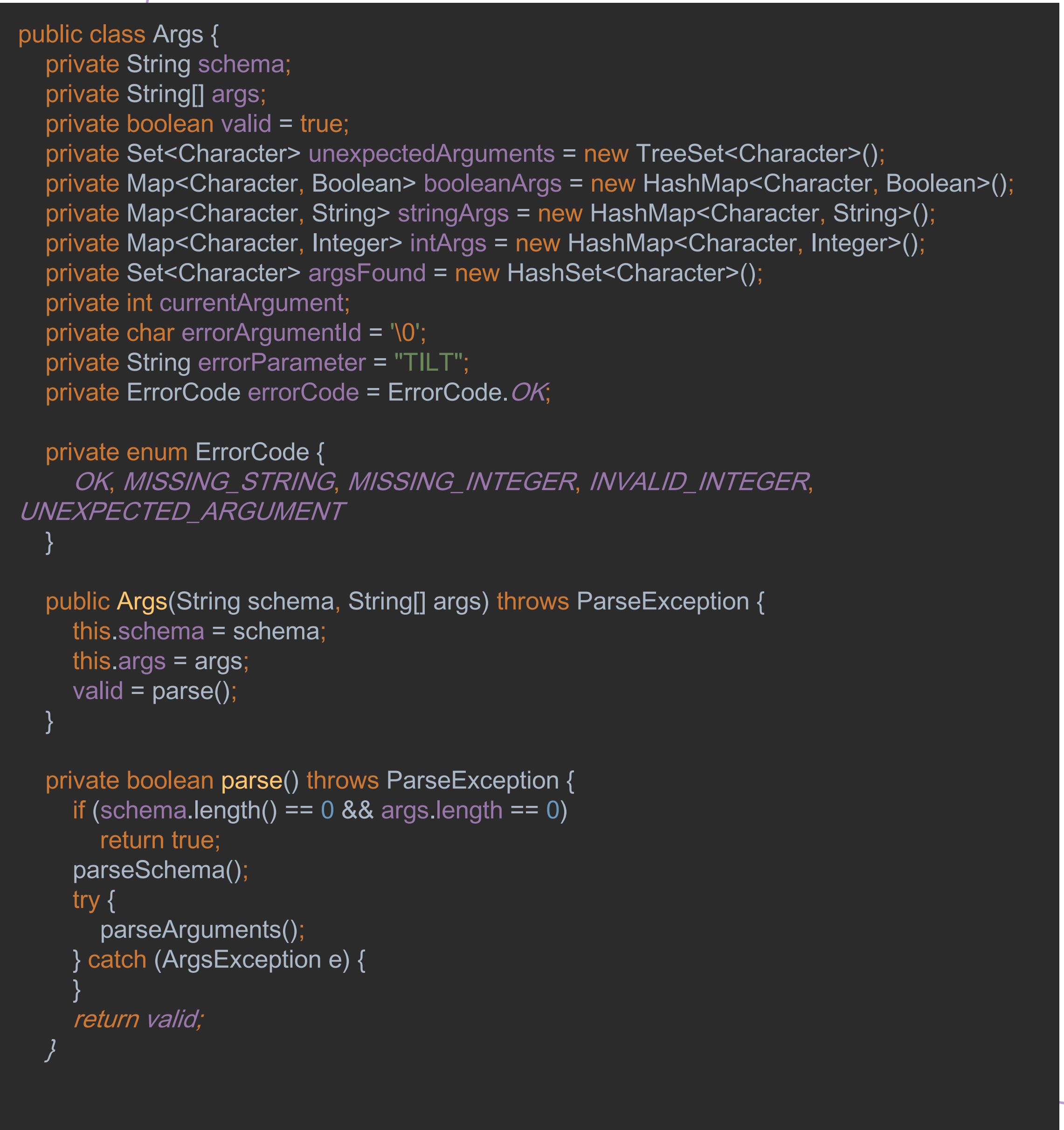
Identifier + Indicator – ex) "l,p#,d*"

Identifier	Indicator	Argument Type
Char	(nothing)	Bool type argument
Char	*	String type argument
Char	#	Integer type argument
Char	##	Double type argument

동작 순서

1. parseSchema
   식별자 구분 & 해당 식별자(Identifier)에 해당하는 타입의 arg를 저장할 변수 초기화
2. parseArguments
   Argument 저장 변수의 멤버 변수 value를 argument String의 해당 부분으로 초기화
   + argFound에 해당 type의 **식별자**를 추가함

1차 초안 Args

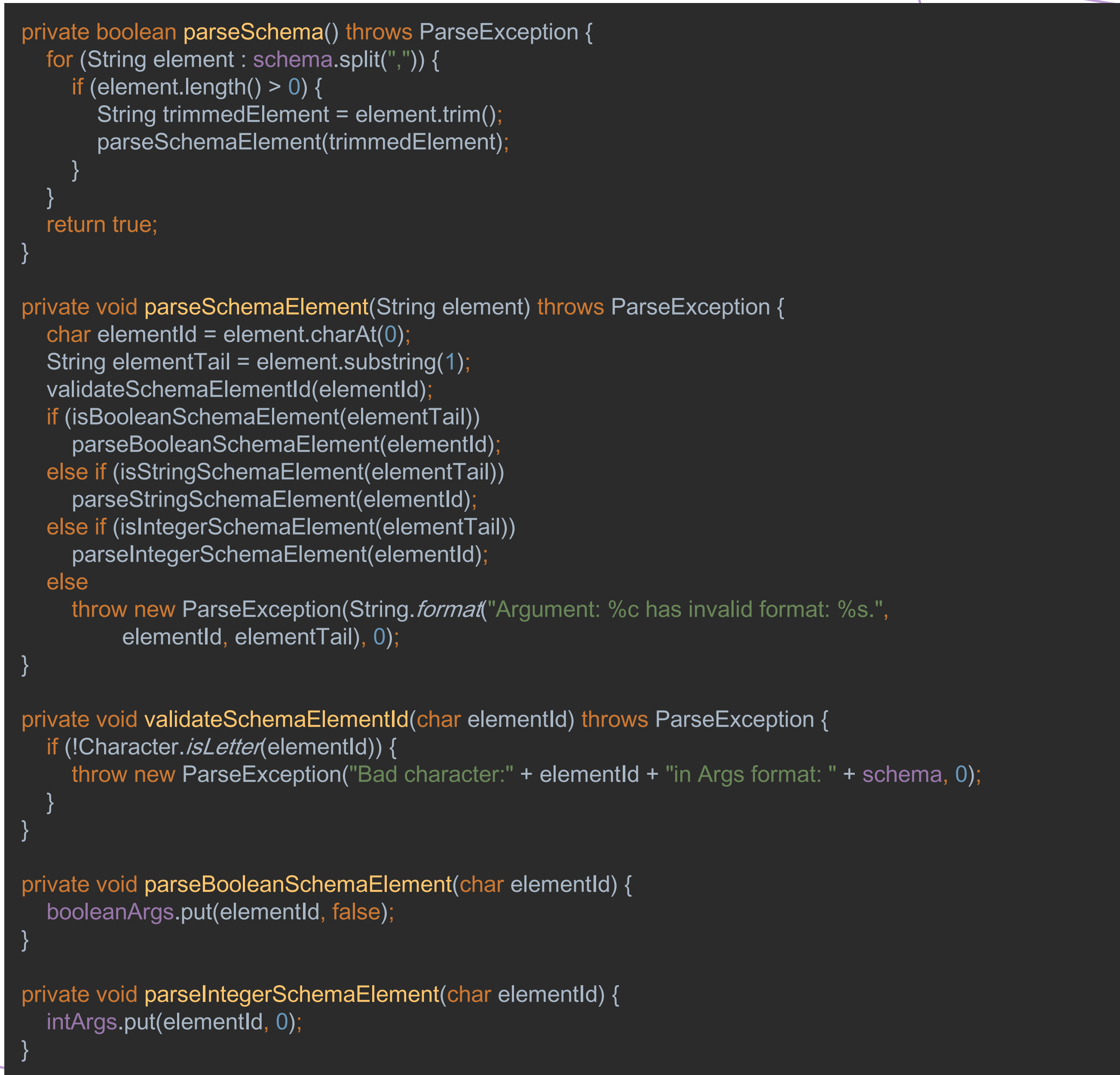


```
public class Args {
    private String schema;
    private String[] args;
    private boolean valid = true;
    private Set<Character> unexpectedArguments = new TreeSet<Character>();
    private Map<Character, Boolean> booleanArgs = new HashMap<Character, Boolean>();
    private Map<Character, String> stringArgs = new HashMap<Character, String>();
    private Map<Character, Integer> intArgs = new HashMap<Character, Integer>();
    private Set<Character> argsFound = new HashSet<Character>();
    private int currentArgument;
    private char errorArgumentId = '\0';
    private String errorParameter = "TILT";
    private ErrorCode errorCode = ErrorCode.OK;

    private enum ErrorCode {
        OK, MISSING_STRING, MISSING_INTEGER, INVALID_INTEGER,
UNEXPECTED_ARGUMENT
    }

    public Args(String schema, String[] args) throws ParseException {
        this.schema = schema;
        this.args = args;
        valid = parse();
    }

    private boolean parse() throws ParseException {
        if (schema.length() == 0 && args.length == 0)
            return true;
        parseSchema();
        try {
            parseArguments();
        } catch (ArgsException e) {
        }
        return valid;
    }
```



```
private boolean parseSchema() throws ParseException {
    for (String element : schema.split(",")) {
        if (element.length() > 0) {
            String trimmedElement = element.trim();
            parseSchemaElement(trimmedElement);
        }
    }
    return true;
}

private void parseSchemaElement(String element) throws ParseException {
    char elementId = element.charAt(0);
    String elementTail = element.substring(1);
    validateSchemaElementId(elementId);
    if (isBooleanSchemaElement(elementTail))
        parseBooleanSchemaElement(elementId);
    else if (isStringSchemaElement(elementTail))
        parseStringSchemaElement(elementId);
    else if (isIntegerSchemaElement(elementTail))
        parseIntegerSchemaElement(elementId);
    else
        throw new ParseException(String.format("Argument: %c has invalid format: %s.",
                elementId, elementTail), 0);
}

private void validateSchemaElementId(char elementId) throws ParseException {
    if (!Character.isLetter(elementId)) {
        throw new ParseException("Bad character:" + elementId + "in Args format: " + schema, 0);
    }
}

private void parseBooleanSchemaElement(char elementId) {
    booleanArgs.put(elementId, false);
}

private void parseIntegerSchemaElement(char elementId) {
    intArgs.put(elementId, 0);
}
```

```java
private void parseStringSchemaElement(char elementId) {
   stringArgs.put(elementId, "");
}

private boolean isStringSchemaElement(String elementTail) {
   return elementTail.equals("*");
}

private boolean isBooleanSchemaElement(String elementTail) {
   return elementTail.length() == 0;
}

private boolean isIntegerSchemaElement(String elementTail) {
   return elementTail.equals("#");
}

private boolean parseArguments() throws ArgsException {
   for (currentArgument = 0; currentArgument < args.length; currentArgument++) {
      String arg = args[currentArgument];
      parseArgument(arg);
   }
   return true;
}

private void parseArgument(String arg) throws ArgsException {
   if (arg.startsWith("-"))
      parseElements(arg);
}

private void parseElements(String arg) throws ArgsException {
   for (int i = 1; i < arg.length(); i++)
      parseElement(arg.charAt(i));
}

private void parseElement(char argChar) throws ArgsException {
   if (setArgument(argChar))
      argsFound.add(argChar);
   else {
      unexpectedArguments.add(argChar);
      errorCode = ErrorCode.UNEXPECTED_ARGUMENT;
      valid = false;
   }
}
```

```java
private boolean setArgument(char argChar) throws ArgsException {
   if (isBooleanArg(argChar))
      setBooleanArg(argChar, true);
   else if (isStringArg(argChar))
      setStringArg(argChar);
   else if (isIntArg(argChar))
      setIntArg(argChar);
   else
      return false;

   return true;
}

private boolean isIntArg(char argChar) {
   return intArgs.containsKey(argChar);
}

private void setIntArg(char argChar) throws ArgsException {
   currentArgument++;
   String parameter = null;
   try {
      parameter = args[currentArgument];
      //noinspection removal
      intArgs.put(argChar, new Integer(parameter));
   } catch (ArrayIndexOutOfBoundsException e) {
      valid = false;
      errorArgumentId = argChar;
      errorCode = ErrorCode.MISSING_INTEGER;
      throw new ArgsException();
   } catch (NumberFormatException e) {
      valid = false;
      errorArgumentId = argChar;
      errorParameter = parameter;
      errorCode = ErrorCode.INVALID_INTEGER;
      throw new ArgsException();
   }
}
```

```java
private boolean isStringArg(char argChar) {
    return stringArgs.containsKey(argChar);
}

private void setBooleanArg(char argChar, boolean value) {
    booleanArgs.put(argChar, value);
}

private boolean isBooleanArg(char argChar) {
    return booleanArgs.containsKey(argChar);
}

public int cardinality() {
    return argsFound.size();
}

public String usage() {
    if (schema.length() > 0)
        return "-[" + schema + "]";
    else
        return "";
}

public String errorMessage() throws Exception {
    switch (errorCode) {
        case OK:
            throw new Exception("TILT: Should not get here.");
        case UNEXPECTED_ARGUMENT:
            return unexpectedArgumentMessage();
        case MISSING_STRING:
            return String.format("Could not find string parameter for -%c.", errorArgumentId);
        case INVALID_INTEGER:
            return String.format("Argument -%c expects an integer but was '%s'.", errorArgumentId, errorParameter);
        case MISSING_INTEGER:
            return String.format("Could not find integer parameter for -%c.", errorArgumentId);
    }
    return "";
}
```

```java
private String unexpectedArgumentMessage() {
        StringBuffer message = new StringBuffer("Argument(s) -");
        for (char c : unexpectedArguments) {
            message.append(c);
        }
        message.append(" unexpected.");

        return message.toString();
    }

    private boolean falseIfNull(Boolean b) {
        return b != null && b;
    }

    private int zeroIfNull(Integer i) {
        return i == null ? 0 : i;
    }

    private String blankIfNull(String s) {
        return s == null ? "" : s;
    }

    public String getString(char arg) {
        return blankIfNull(stringArgs.get(arg));
    }

    public int getInt(char arg) {
        return zeroIfNull(intArgs.get(arg));
    }

    public boolean getBoolean(char arg) {
        return falseIfNull(booleanArgs.get(arg));
    }

    public boolean has(char arg) {
        return argsFound.contains(arg);
    }

    public boolean isValid() {
        return valid;
    }

    private class ArgsException extends Exception{

    }
```

최종 Args

```java
public class Args {
    private Map<Character, ArgumentMarshaler> marshalers;
    private Set<Character> argsFound;
    private ListIterator<String> currentArgument;

    public Args(String schema, String[] args) throws ArgsException{
        marshalers = new HashMap<Character, ArgumentMarshaler>();
        argsFound = new HashSet<Character>();

        parseSchema(schema);
        parseArgumentStrings(Arrays.asList(args));
    }


    private void parseSchema(String schema) throws ArgsException{
        for( String element: schema.split(","))
            if(element.length() > 0)
                parseSchemaElement(element.trim());
    }

    private void parseSchemaElement(String element) throws ArgsException{
        char elementId = element.charAt(0);
        String elementTail = element.substring(1);
        validateSchemaElementId(elementId);
        if(elementTail.length() == 0)
            marshalers.put(elementId, new BooleanArgumentMarshaler());
        else if(elementTail.equals("*"))
            marshalers.put(elementId, new StringArgumentMarshaler());
        else if(elementTail.equals("#"))
            marshalers.put(elementId, new IntegerArgumentMarshaler());
        else if(elementTail.equals("##"))
            marshalers.put(elementId, new DoubleArgumentMarshaler());
        else if(elementTail.equals("[*]"))
            marshalers.put(elementId, new StringArrayArgumentMarshaler());
        else if (elementTail.equals("&"))
            marshalers.put(elementId, new MapArgumentMarshaler());
        else
            throw new ArgsException(INVALID_ARGUMENT_FORMAT, elementId, elementTail);
    }
```

```java
private void validateSchemaElementId(char elementId) throws ArgsException{
    if(!Character.isLetter(elementId))
        throw new ArgsException(INVALID_ARGUMENT_NAME, elementId, null);
}

private void parseArgumentStrings(List<String> argsList) throws ArgsException {
    for(currentArgument = argsList.listIterator(); currentArgument.hasNext();){
        String argString = currentArgument.next();
        if(argString.startsWith("-")){
            parseArgumentCharacters(argString.substring(1));
        }else{
            currentArgument.previous();
            break;
        }
    }
}

private void parseArgumentCharacters(String argChars) throws ArgsException {
    for (int i=0;i< argChars.length();i++)
        parseArgumentCharacter(argChars.charAt(i));

}

private void parseArgumentCharacter(char argChar) throws ArgsException{
    ArgumentMarshaler m = marshalers.get(argChar);
    if(m==null){
        throw new ArgsException(UNEXPECTED_ARGUMENT, argChar, null);
    }else{
        argsFound.add(argChar);
        try{
            m.set(currentArgument);
        }catch (ArgsException e){
            e.setErrorArgumentId(argChar);
            throw e;
        }
    }
}

public boolean has(char arg){
    return argsFound.contains(arg);
}
```

```java
public int nextArgument(){
      return currentArgument.nextIndex();
   }

   public boolean getBoolean(char arg){
      return BooleanArgumentMarshaler.getValue(marshalers.get(arg));
   }

   public String getString(char arg){
      return StringArgumentMarshaler.getValue(marshalers.get(arg));
   }

   public int getInt(char arg){
      return IntegerArgumentMarshaler.getValue(marshalers.get(arg));
   }

   public double getDouble(char arg){
      return DoubleArgumentMarshaler.getValue(marshalers.get(arg));
   }

   public String[] getStringArray(char arg){
      return StringArrayArgumentMarshaler.getValue(marshalers.get(arg));
   }

   public Map<String, String> getMap(char arg) {
      return MapArgumentMarshaler.getValue(marshalers.get(arg));
   }
}
```

```java
public class IntegerArgumentMarshaler implements ArgumentMarshaler {
   private int intValue = 0;

   public void set(Iterator<String> currentArgument) throws ArgsException {
      String parameter = null;
      try {
         parameter = currentArgument.next();
         intValue = Integer.parseInt(parameter);
      } catch (NoSuchElementException e) {
         throw new ArgsException(MISSING_INTEGER);
      } catch (NumberFormatException e) {
         throw new ArgsException(INVALID_INTEGER, parameter);
      }
   }

   public static int getValue(ArgumentMarshaler am) {
      if (am != null && am instanceof IntegerArgumentMarshaler)
         return ((IntegerArgumentMarshaler) am).intValue;
      else
         return 0;
   }
}
```

테스트 코드 확인하기 - JUnit