National University
Manila

ARMADA: A High-Integrity Programming Language for Developing Security-Driven Systems

In Partial Fulfillment of Requirement in
Programming Languages

Submitted by:
Daskeo, Kristine M.
Manalo, John Christian A.
Ramos, Julianne Adrielle T.

COM211

Professor:
Ms. Armida P. Salazar

College of Computing and Information Technologies
Computer Science Department

November 5, 2024

I. INTRODUCTION

A. DESCRIPTION

ARMADA, which means a fleet of warships or a large force or group usually of moving things, is created with a mission: to support the development of reliable and high-integrity systems. It is an ideal choice for industries where reliability and security are paramount, such as aerospace, automotive, and healthcare, where ensuring the correctness and stability of systems.

PROGRAM GRAMMAR:

```
<program> → <statements>
<statements> → <statement> | <statement><statements>
<statement> → <coords-type> | <mach-expression> | <print-function> |
                  <case-conditional> | <create-declaration>
```

II. ARMADA PROGRAMMING CONSTRUCTS

1. VARIABLE DECLARATIONS

1.1. Coordinate Values

Description:

Coords is short for ‘coordinates’ which can store latitude, longitude, and altitude values to provide 3D positioning of an aircraft. Latitude and longitude will follow decimal degree (DD) notation where positive and negative signs indicate direction. Altitude is measured by feet above sea level.

Sample Syntax:

```
coords location := (14.599512, 120.984222, 30000);
```

CFG Definition:

```
<coords-type> → <type> <identifier> <assign-operator>
                  <open-separator><double-value><value-separator>
                  <double-value><value-separator><long-value>
                  <close-separator><terminator>

<type> → "coords”

<identifier> → <string>
<string> → <char> | <char><string>
<char> → <letter> | <digit> | “-”
<letter> → <lowercase> | <uppercase>
```

```
<lowercase> → "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j"
| "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v"
| "w" | "x" | "y" | "z"
<uppercase> → "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J"
| "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V"
| "W" | "X" | "Y" | "Z"
<digit> → "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

<assign-operator> → “:=”

<open-separator> → "("
<close-separator> → ")"

<value-separator> → ","

<double-value> → [±]<digits> | [±]<digits><point-separator><digits>

<point-separator> → "."

<long-value> → <digits>

<digits> → <digit> | <digit> <digits>

<digit> → "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

<terminator> → ";"
```

1.2. Aircraft Status

Description:
Aircraft status defines the aviation’s state—whether departed, airborne, or landed. Departed specifies that the airplane is currently taxiing and about to takeoff. Airborne defines the plane as currently enroute to its station. Landed indicates that it has arrived at its destination airport.

Sample Syntax:
```
status flightStatus;
flightStatus := “Landed”;
```

CFG Definition:
```
<status-type> → <status-declaration> <status-assignment>
```

```
<status-declaration> → <type> <identifier><terminator>
<status-assignment> → <identifier> <assign-operator>
                        <quote-separator><status>
                        <quote-separator><terminator>

<type> → “status”

<identifier> → <string>
<string> → <char> | <char><string>
<char> → <letter> | <digit> | “-”
<letter> → <lowercase> | <uppercase>
<lowercase> → "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j"
| "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v"
| "w" | "x" | "y" | "z"
<uppercase> → "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J"
| "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V"
| "W" | "X" | "Y" | "Z"
<digit> → "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

<assign-operator> → “:=”

<quote-separator> → ‘“‘

<status> → “Boarding” | “Airborne” | “Landed”

<terminator> → ";"
```

2. Expressions

2.1. Mach

Description:
One way to measure an aircraft’s speed is by calculating its Mach number, or simply Mach. It represents an airplane's speed compared with the speed of sound in air. Mathematically, it is defined as the local flow velocity over the speed of sound in the medium.

Syntax:
```
double speed := Mach(205.78, 328.3);
print(speed);
```

Output:
0.626

CFG Definition:

```
<mach-expression> → <data-type> <identifier> <assign-operator>
                    <keyword><open-separator><double-value>
                    <value-separator><double-value><close-separator>
                    <terminator>

<data-type> → “double”

<identifier> → <string>
<string> → <char> | <char><string>
<char> → <letter> | <digit> | “-”
<letter> → <lowercase> | <uppercase>
<lowercase> → "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j"
| "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v"
| "w" | "x" | "y" | "z"
<uppercase> → "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J"
| "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V"
| "W" | "X" | "Y" | "Z"
<digit> → "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

<assign-operator> → “:=”

<keyword> → “Mach”

<open-separator> → “(“
<close-separator> → “)”

<value-separator> → “,”

<double-value> → [±]<digits> | [±]<digits><point-separator><digits>
<digits> → <digit> | <digit> <digit>
<digit> → "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

<terminator> → ";"
```

3. Function

3.1. Print

Description:
Most, if not all, programming languages implement a function to display information on the user's screen or other standard output device. Similarly, this print function displays a series of characters in a line of text.

Sample Syntax:

```
print("Hello World!");

coords location := (14.599512, 120.984222, 30000);
print(location);
```

```
Output:
Hello World!
Coords(14.599512, 120.984222, 30000)
```

CFG Definition:

```
<print-function> → <keyword><open-separator><print-stmt>
                   <close-separator><terminator>

<print-stmt> → <print-string> | <print-identifier>

<print-string> → <quote-separator><string><quote-separator>

<print-identifier> → <string>

<keyword> → "print"

<open-separator> → "("
<close-separator> → ")"

<string> → <char> | <char><string>
<char> → <letter> | <digit> | "-"
<letter> → <lowercase> | <uppercase>
<lowercase> → "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j"
| "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v"
| "w" | "x" | "y" | "z"
<uppercase> → "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J"
| "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V"
| "W" | "X" | "Y" | "Z"
```

```
<digit> → "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

<terminator> → ";"
```

4. Conditionals

4.1. Case

Description:
Case is a conditional construct that runs a controlled block of code in an instance where the specified stipulations are met. It only works by comparing two values through different logical operations. This can be used to change an airplane's status if its stored location matches a specific coordinate.

Syntax:

```
case (a == b) {
      // valid
}
case (x > 10) {
      // valid
}
case (x !=) {
      // invalid condition
}
case (value >= 20) {
      // valid
}
case (temp <) {
      // invalid condition
}
```

CFG Definition:

```
<case-conditional> → <keyword> <open-separator><conditions>
                        <close-separator><open-brace><statements>
                        <close-brace>

<keyword> → “create”

<open-separator> → “(“
<close-separator> → “)”

<conditions> → <simple-condition> | <compound-condition>
```

```
<simple-condition> → <identifier> <operator> <identifier>

<compound-condition> → <simple-condition> ( <logical-operator>
                        <simple-condition> )*

<operator> → "==" | "!=" | ">" | "<" | ">=" | "<="

<logical-operator> → "&&" | "||"

<identifier> → <string>
<string> → <char> | <char><string>
<char> → <letter> | <digit> | “-”
<letter> → <lowercase> | <uppercase>
<lowercase> → "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j"
| "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v"
| "w" | "x" | "y" | "z"
<uppercase> → "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J"
| "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V"
| "W" | "X" | "Y" | "Z"
<digit> → "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

<terminator> → “;”
```

5. Objects

5.1. Create

Description:
Objects are part of the building blocks in Object-Oriented Programming (OOP). Create is a way to initialize an object in Armada which can be utilized to define an aircraft class and its attributes.

Syntax:

```
create Airplane {
      string id;
      string name;
      coords location;
      status flightStatus;
}
```

CFG Definition:

```
<create-declaration> → <keyword> <identifier><open-brace><fields>
                       <close-brace>

<keyword> → “create”

<open-brace> → “{“
<close-brace> → “}“
<fields> → <field> | <field><fields>
<field> → <data-type> <identifier><terminator>

<data-type> → “string” | “coords” | “status”

<identifier> → <string>
<string> → <char> | <char><string>
<char> → <letter> | <digit> | “-”
<letter> → <lowercase> | <uppercase>
<lowercase> → "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j"
| "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v"
| "w" | "x" | "y" | "z"
<uppercase> → "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J"
| "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V"
| "W" | "X" | "Y" | "Z"
<digit> → "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

<terminator> → “;”
```

III. SYNTAX CHECKER INSTRUCTIONS

The syntax checker for ARMADA is distributed among three (3) java files. In a Java compiler, run the specific class based on the construct the user wants to verify.

- **`Main.java`** handles the checking for **`coords`** declaration, **`Mach`** expression, and **`print`** function.
- **`IfCFG.java`** is for checking **`case`** conditionals.
- **`ObjectCFG.java`** deals with the checking of object creation.

CFG Definition:

```
<Main> → <coords-type> | <mach-expression> | <print-function>
<IfCFG> → <case-conditional>
<ObjectCFG> → <create-declaration>
```

In order to initiate the syntax check, type ` `END` ` at the end of the code block.

IV. TEST CASES

1. coord (MainCFG.java)

Valid Input:

```
coords Location := (14.5123, 121.0665, 13);
END
```

Expected Output:

```
Location of type coordinates is set to (latitude = 14.5123, longitude = 121.0665, altitude = 13)
```

```
Enter your code (type 'END' on a new line to finish):
coords Location := (14.5123, 121.0665, 13);
END
Location of type coordinates is set to (latitude = 14.5123, longitude = 121.0665, altitude = 13)
```

Invalid Input:

```
Coords Location := (14.5123, 121.0665, 13);
END
```

Expected Output:

```
Error: Invalid syntax -> Coords Location := (14.5123, 121.0665, 13);
Check for missing or misplaced semi-colons and correct variable names.
```

```
Enter your code (type 'END' on a new line to finish):
Coords Location := (14.5123, 121.0665, 13);
END
Error: Invalid syntax -> Coords Location := (14.5123, 121.0665, 13);
Check for missing or misplaced semi-colons and correct variable names.
```

Invalid Input:

```
coords loc := (131.41221, 413123.43, 131.31);
END
```

Expected Output:

```
Error: Wrong number format. It must be (double latitude, double longitude, long altitude).
```

```
Enter your code (type 'END' on a new line to finish):
coords loc := (131.41221, 413123.43, 131.31);
END
Error: Wrong number format. It must be (double latitude, double longitude, long altitude).
```

Invalid Input:

```
coords loc := (14.000, 120.000);
END
```

Expected Output:

Error: Missing value. It must be (double latitude, double longitude, long altitude).

```
Enter your code (type 'END' on a new line to finish):
coords loc := (14.000, 120.000);
END
Error: Missing value. It must be (double latitude, double longitude, long altitude).
```

2. **Mach (MainCFG.java)**

Valid Input:

```
double Speed := Mach(25.0, 5.0);
END
```

Expected Output:

Speed is equal to 5.0

```
Enter your code (type 'END' on a new line to finish):
double Speed := Mach(25.0, 5.0);
END
Speed is equal to 5.0
```

Invalid Input:

```
double speed := Mach(21);
END
```

Expected Output:

Error: Missing values for Mach. It must be (double value1, double value2).

```
Enter your code (type 'END' on a new line to finish):
double speed := Mach(21);
END
Error: Missing values for Mach. It must be (double value1, double value2).
```

Invalid Input:

```
long Speed := Mach(25.0, 5.0);
END
```

Expected Output:

Error: Invalid syntax -> long Speed := Mach(25.0, 5.0);
Check for missing or misplaced semi-colons and correct variable names.

```
Enter your code (type 'END' on a new line to finish):
long Speed := Mach(25.0, 5.0);
END
Error: Invalid syntax -> long Speed := Mach(25.0, 5.0);
Check for missing or misplaced semi-colons and correct variable names.
```

3. **print (MainCFG.java)**

Valid Input:

```
print("Hello World!");
END
```

Expected Output:

```
Hello World!
```

```
Enter your code (type 'END' on a new line to finish):
print("Hello World!");
END
Hello World!
```

Valid Input:

```
coords Location := (14.5123, 121.0665, 13);
print(Location);
END
```

Expected Output:

```
Location of type coordinates is set to (latitude = 14.5123, longitude =
121.0665, altitude = 13)
(latitude = 14.5123, longitude = 121.0665, altitude = 13)
```

```
Enter your code (type 'END' on a new line to finish):
coords Location := (14.5123, 121.0665, 13);
print(Location);
END
Location of type coordinates is set to (latitude = 14.5123, longitude = 121.0665, altitude = 13)
(latitude = 14.5123, longitude = 121.0665, altitude = 13)
```

Invalid Input:

```
Print(“Hello World!”);
END
```

Expected Output:

```
Error: Invalid syntax -> Print(“Hello World!”);
Check for missing or misplaced semi-colons and correct variable names.
```

```
Enter your code (type 'END' on a new line to finish):
Print(“Hello World!”);
END
Error: Invalid syntax -> Print(“Hello World!”);
Check for missing or misplaced semi-colons and correct variable names.
```

Invalid Input:

```
print(location);
END
```

Expected Output:

```
Error: Identifier 'location' not found. Check variable declarations.
```

```
Enter your code (type 'END' on a new line to finish):
print(location);
END
Error: Identifier 'location' not found. Check variable declarations.
```

Invalid Input:

```
print (“Hello World!”);
END
```

Expected Output:

```
Error: Invalid syntax -> print (“Hello World!”);
Check for missing or misplaced semi-colons and correct variable names.
```

```
Enter your code (type 'END' on a new line to finish):
print (“Hello World!”);
END
Error: Invalid syntax -> print (“Hello World!”);
Check for missing or misplaced semi-colons and correct variable names.
```

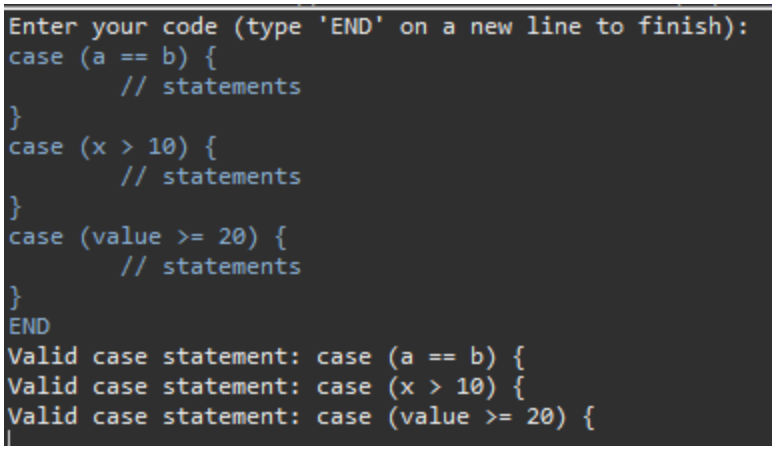
4. **case (IfCFG.java)**

Valid Input:

```
case (a == b) {
      // statements
}
case (x > 10) {
      // statements
}
case (value >= 20) {
      // statements
}
END
```

Expected Output:

```
Valid case statement: case (a == b) {
Valid case statement: case (x > 10) {
Valid case statement: case (value >= 20) {
```



Valid Input:

```
status flightStatus;
coords location := (0, 0, 0);
coords NAIA := (14.5123, 121.0165, 13);
coords DIA := (7.122552, 125.64550, 22);

case (location == NAIA) {
      flightStatus := "Landed";
}
case (location != NAIA && location != DIA) {
      flightStatus := "Airborne";
}
case (location == DIA) {
      flightStatus := "Boarding";
}
END
```

Expected Output:

```
Data type declared flightStatus of type status.
location of type coordinates is set to (latitude = 0.0, longitude = 0.0,
altitude = 0)
NAIA of type coordinates is set to (latitude = 14.5123, longitude =
121.0165, altitude = 13)
DIA of type coordinates is set to (latitude = 7.122552, longitude =
125.6455, altitude = 22)
Valid case statement: case (location == NAIA) {
flightStatus of type status is set to Landed.
Valid case statement: case (location != NAIA && location != DIA) {
flightStatus of type status is set to Airborne.
Valid case statement: case (location == DIA) {
```

flightStatus of type status is set to Boarding.

```
Enter your code (type 'END' on a new line to finish):
status flightStatus;
coords location := (0, 0, 0);
coords NAIA := (14.5123, 121.0165, 13);
coords DIA := (7.122552, 125.64550, 22);

case (location == NAIA) {
        flightStatus := "Landed";
}
case (location != NAIA && location != DIA) {
        flightStatus := "Airborne";
}
case (location == DIA) {
        flightStatus := "Boarding";
}
END
Data type declared flightStatus of type status.
location of type coordinates is set to (latitude = 0.0, longitude = 0.0, altitude = 0)
NAIA of type coordinates is set to (latitude = 14.5123, longitude = 121.0165, altitude = 13)
DIA of type coordinates is set to (latitude = 7.122552, longitude = 125.6455, altitude = 22)
Valid case statement: case (location == NAIA) {
flightStatus of type status is set to Landed.
Valid case statement: case (location != NAIA && location != DIA) {
flightStatus of type status is set to Airborne.
Valid case statement: case (location == DIA) {
flightStatus of type status is set to Boarding.
```

Invalid Input:

```
case (x != ) {
      // stmts
}
case (temp <) {
      // stmts
}
END
```

Expected Output:

```
Invalid case statement (invalid condition): case (x != ) {
Invalid case statement (invalid condition): case (temp <) {
```

```
Enter your code (type 'END' on a new line to finish):
case (x != ) {
        // stmts
}
case (temp <) {
        // stmts
}
END
Invalid case statement (invalid condition): case (x != ) {
Invalid case statement (invalid condition): case (temp <) {
```

Invalid Input:

```
case (a == b) {
END
```

Expected Output:

```
Invalid case statement (invalid condition): case (x != ) {
Invalid case statement (invalid condition): case (temp <) {
```

```
Enter your code (type 'END' on a new line to finish):
case (a == b) {
END
Invalid case statement (missing closing brace): case (a == b) {
```

Invalid Input:

```
case (location) {
}
END
```

Expected Output:

```
Invalid case statement (invalid condition): case (location) {
```

```
Enter your code (type 'END' on a new line to finish):
case (location) {
}
END
Invalid case statement (invalid condition): case (location) {
```

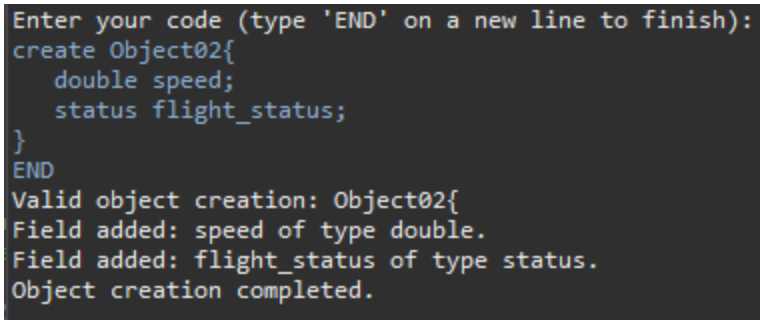
5. **create (ObjectCFG.java)**

Valid Input:

```
create Object02{
      double speed;
      status flight_status;
}
END
```

Expected Output:

```
Valid object creation: Object02{
Field added: speed of type double.
Field added: flight_status of type status.
Object creation completed.
```



Valid Input:

```
create Object {
      string name;
      coords location;
      double speed;
      status flightStatus;
}
Object airplane;
airplane.name := "Boeing";
airplane.location := (14.5123, 121.0165, 13);
airplane.speed := Mach(25.0, 5.0);
airplane.flightStatus := "Landed";
print(airplane.name);
print(airplane.location);
print(airplane.speed);
END
```

Expected Output:

```
Valid object creation: Object
Field added: name of type string.
Field added: location of type coords.
Field added: speed of type double.
Field added: flightStatus of type status.
Object creation completed.
Object instance created: airplane of type Object
Assigned string to airplane.name
Assigned coords to airplane.location
Assigned Mach result to airplane.speed
Assigned status to airplane.flightStatus
Field value for airplane.name: Boeing
Field value for airplane.location: Coordinates (latitude=14.5123,
longitude=121.0165, altitude=13)
Field value for airplane.speed: 125.0
```

```
Enter your code (type 'END' on a new line to finish):
create Object {
        string name;
        coords location;
        double speed;
        status flightStatus;
}
Object airplane;
airplane.name := "Boeing";
airplane.location := (14.5123, 121.0165, 13);
airplane.speed := Mach(25.0, 5.0);
airplane.flightStatus := "Landed";
print(airplane.name);
print(airplane.location);
print(airplane.speed);
END
Valid object creation: Object
Field added: name of type string.
Field added: location of type coords.
Field added: speed of type double.
Field added: flightStatus of type status.
Object creation completed.
Object instance created: airplane of type Object
Assigned string to airplane.name
Assigned coords to airplane.location
Assigned Mach result to airplane.speed
Assigned status to airplane.flightStatus
Field value for airplane.name: Boeing
Field value for airplane.location: Coordinates (latitude=14.5123, longitude=121.0165, altitude=13)
Field value for airplane.speed: 125.0
```

Invalid Input:

```
create Object {
      string name;
      coords location;
      double speed;
      status flightStatus;
END
```

Expected Output:

```
Valid object creation: Object
Field added: name of type string.
Field added: location of type coords.
Field added: speed of type double.
Field added: flightStatus of type status.
Error: Object creation incomplete, missing closing brace.
```

```
Enter your code (type 'END' on a new line to finish):
create Object {
        string name;
        coords location;
        double speed;
        status flightStatus;
END
Valid object creation: Object
Field added: name of type string.
Field added: location of type coords.
Field added: speed of type double.
Field added: flightStatus of type status.
Error: Object creation incomplete, missing closing brace.
```

Invalid Input:

```
airplane.name := "Boeing";
airplane.location := (14.5123, 121.0165, 13);
airplane.speed := Mach(25.0, 5.0);
airplane.flightStatus := "Landed";
END
```

Expected Output:

```
Error: Object airplane not found.
Error: Object airplane not found.
Error: Object airplane not found.
Error: Object airplane not found.
```

```
Enter your code (type 'END' on a new line to finish):
airplane.name := "Boeing";
airplane.location := (14.5123, 121.0165, 13);
airplane.speed := Mach(25.0, 5.0);
airplane.flightStatus := "Landed";
END
Error: Object airplane not found.
Error: Object airplane not found.
Error: Object airplane not found.
Error: Object airplane not found.
```

Invalid Input:

```
create Object {
      status flightStatus;
}
Object airplane;
airplane.flightStatus := "Delayed";
END
```

Expected Output:

```
Valid object creation: Object
Field added: flightStatus of type status.
Object creation completed.
Object instance created: airplane of type Object
Error: Invalid status -> "Delayed". Valid statuses are 'Landed', 'Airborne', and 'Boarding'.
```