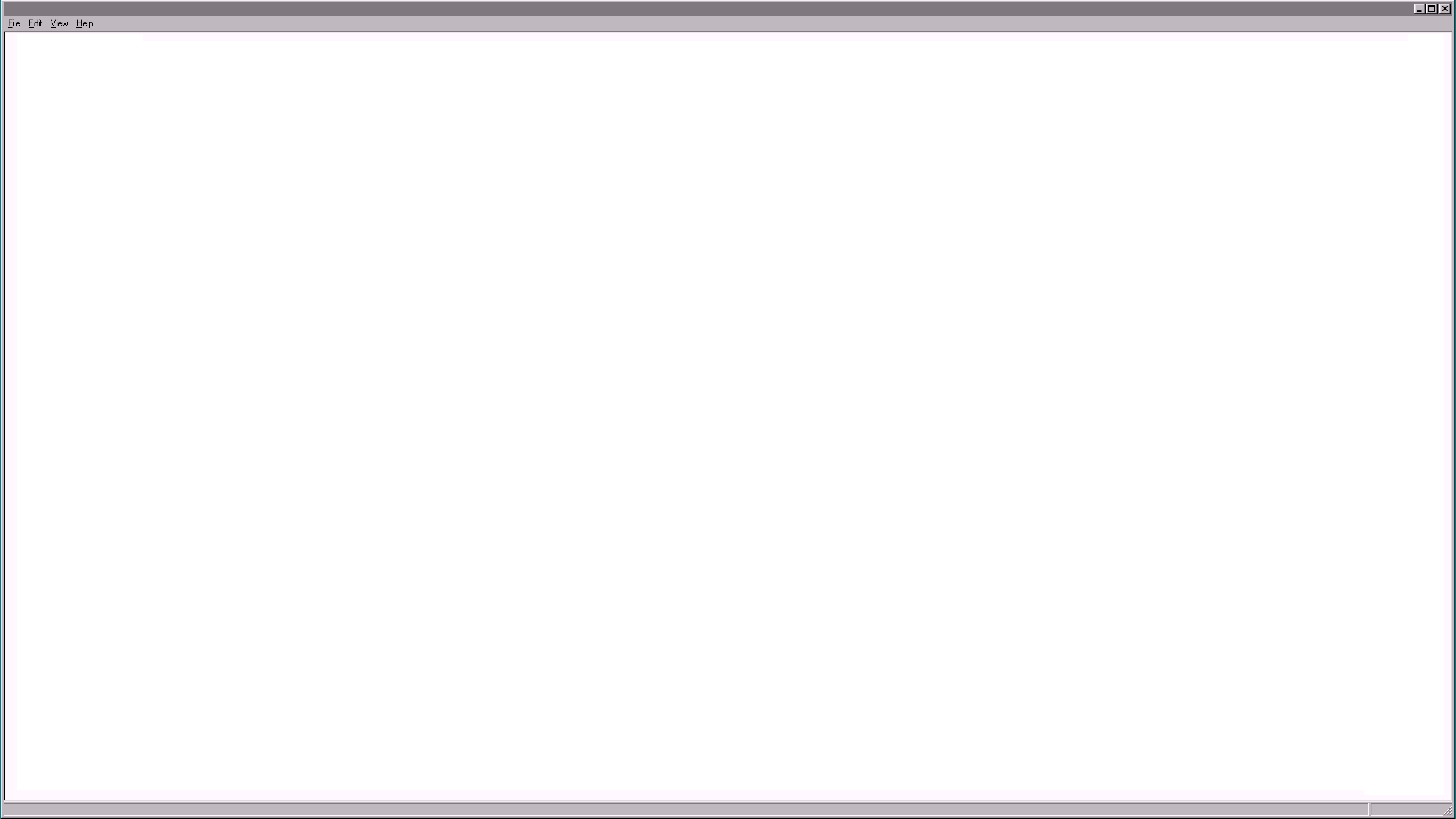
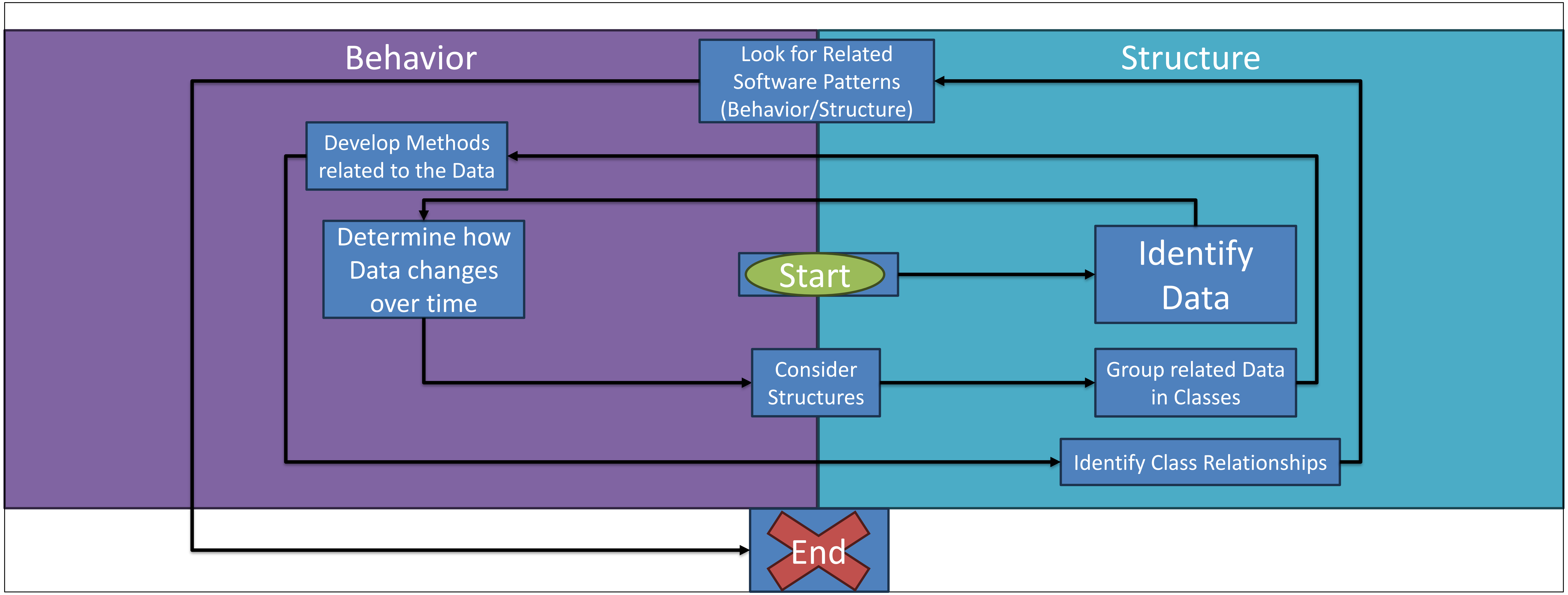

File
Edit
View
Help

File Edit View Help

Programming Review
Part 03

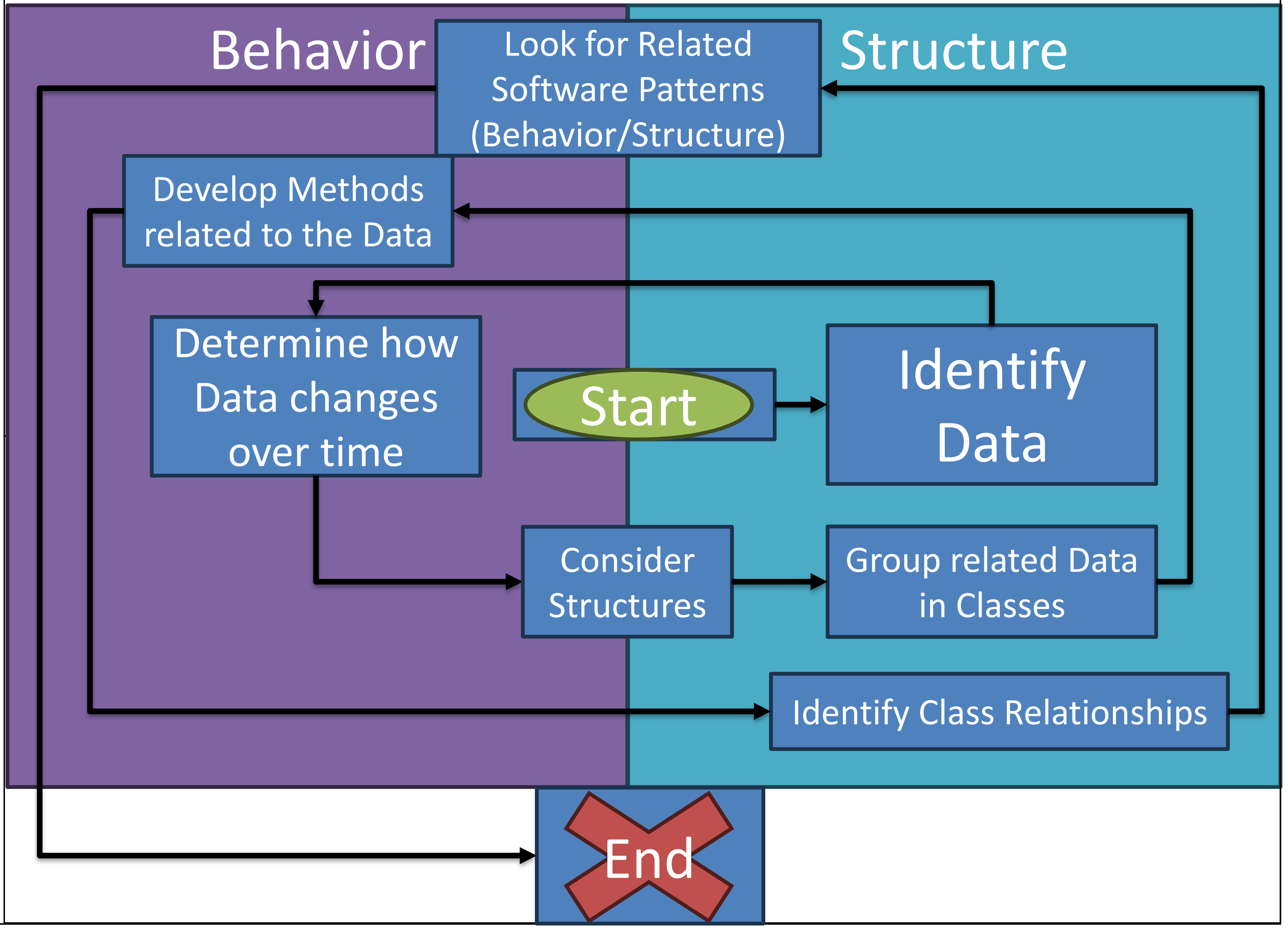
Problem Solving



Behavior
Structure
Look for Related Software Patterns (Behavior/Structure)
Develop Methods related to the Data
Determine how Data changes over time
Start
Identify Data
Consider Structures
Group related Data in Classes
Identify Class Relationships
End

Problem Solving

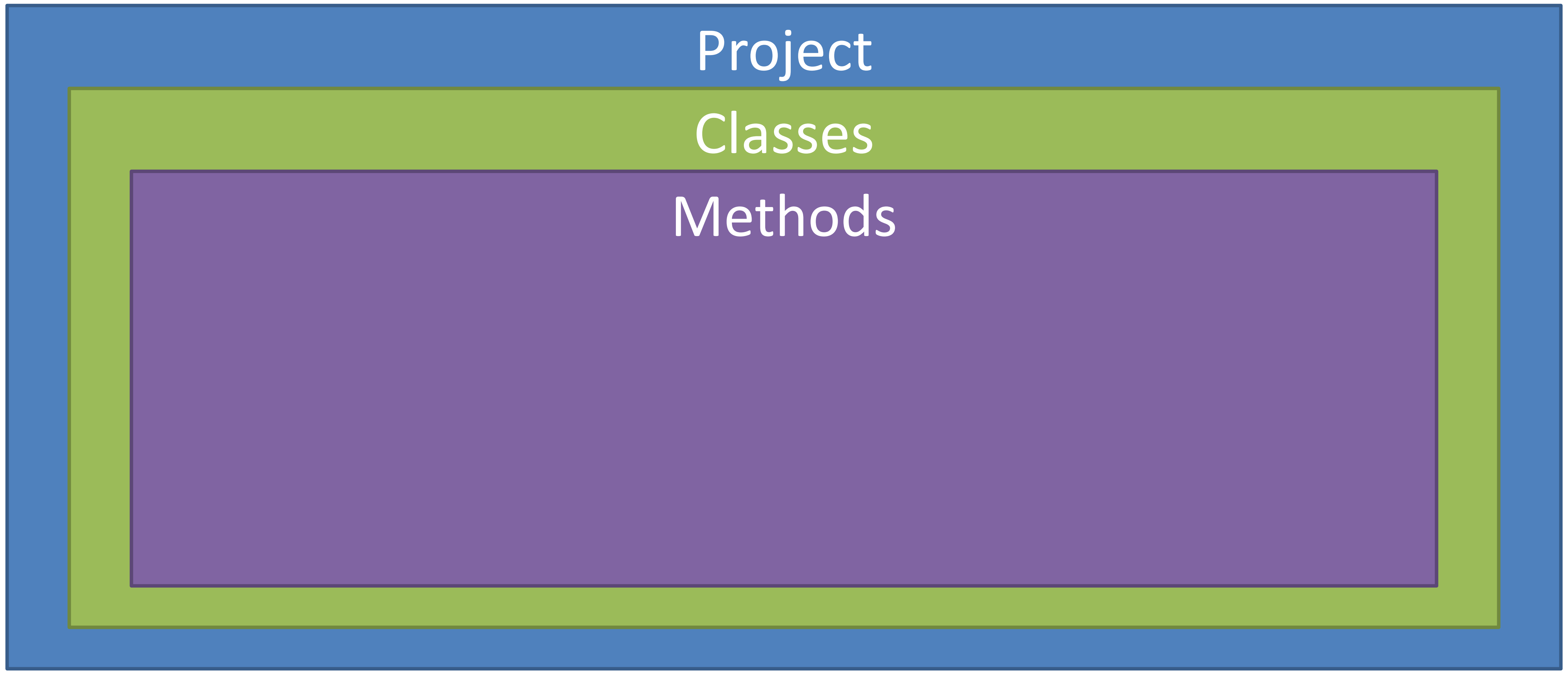
1. Identify your Data.
2. Determine how the data changes over time.
3. Consider structures for both behavior and data.
4. Group together (encapsulate) related information into Classes of Objects.
5. Develop functionality / methodologies that relates to the behavior of your Objects.
6. Further identify relationships between the Classes and optimize the structure.
7. Determine if there exists software patterns that may assist.

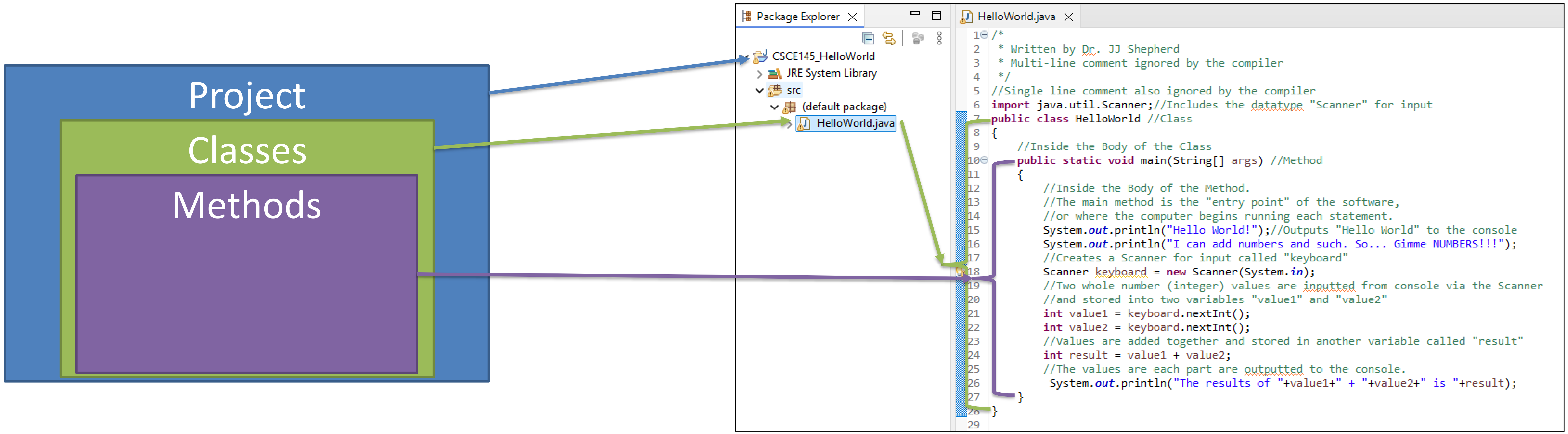

Behavior
Structure
Look for Related Software Patterns (Behavior/Structure)
Develop Methods related to the Data
Determine how Data changes over time
Start
Identify Data
Consider Structures
Group related Data in Classes
Identify Class Relationships
End

Code Organization

Methods and Object-Oriented Programming

File Edit View Help





```
/*
 * Written by Dr. JJ Shepherd
 * Multi-line comment ignored by the compiler
 */
//Single line comment also ignored by the compiler
import java.util.Scanner;//Includes the datatype "Scanner" for input
public class HelloWorld //Class
{
    //Inside the Body of the Class
    public static void main(String[] args) //Method
    {
        //Inside the Body of the Method.
        //The main method is the "entry point" of the software,
        //or where the computer begins running each statement.
        System.out.println("Hello World!");//Outputs "Hello World" to the console
        System.out.println("I can add numbers and such. So... Gimme NUMBERS!!!");
        //Creates a Scanner for input called "keyboard"
        Scanner keyboard = new Scanner(System.in);
        //Two whole number (integer) values are inputted from console via the Scanner
        //and stored into two variables "value1" and "value2"
        int value1 = keyboard.nextInt();
        int value2 = keyboard.nextInt();
        //Values are added together and stored in another variable called "result"
        int result = value1 + value2;
        //The values are each part are outputted to the console.
         System.out.println("The results of "+value1+" + "+value2+" is "+result);
    }
}
```

File Edit View Help

Methods

Syntax for Declaring Dynamic Methods

```
<<scope>> <<return type>> <<identifier>> (<<parameter(s)>>, …)
{
    <<Body of the Method>>
}
```

- Groups functionality into a “callable” structure
- “Verbs”
- Create methods based on singular verbs
- Dynamically created during runtime
- Methods in Java must be *declared* within a Class

Example

```
public boolean isValid(int index)
{
   return index >= 0 && index <a.length;
}
```

Scope and Return Type

- **Scope** indicates *where* the method can be called
  - Public => called outside of the class.
  - Private => only called inside of the class.
- **Return Type** allows methods to pass back values outside of the method
- The “void” return type indicates the method returns nothing
- Any non-void type must return that type of value
  - Must use the word “return” followed by the value
  - Return immediately exits the method
  - All paths must return a value

<u>Example</u>

```
public int getValueFrom(int index)
{
  if(!isValid(index))
    return -1;
  else
    return a[index];
}
private boolean isValid(int i)
{
  return i >= 0 && i <a.length;
}
```

Identifiers and Parameters

- **Identifiers** are the name given to the method
  - Same rules as Variables
  - Good programming practice to "Camel Case" these as well
  - Good programming practice to give method "verb-like" names
- **Parameters** allow outside information to be passed into the method
  - Act as variables for these external values
  - A parameter's scope is only within the body of the method
  - Every parameter needs to be declared (type and id) and separated using a comma ",".

Example

```
public void printMax(int[] a)
{
  if(a == null)//Does "a" exist?
    return;//If not, then leave
  int max = a[0];//Assume first value is max
  for(int i=1;i<a.length;i++)
    max = getMax(max,a[i]);
  System.out.println("Max Value is "+max);
}
private int getMax(int val1, int val2)
{
   if(val1 >= val2)
      return val1;
   else
      return val2;
}
```

Calling Methods

- Using or "calling" methods depends on where it is being called
- Inside the class where it was defined
  - Use the identifier followed by the parameters
  - For dynamic methods it is good practice to use the reserved word "this"
- Outside the class where it was defined
  - An instance of the class (an Object) must be constructed and if not NullPointerException
  - Use the instance followed by a dot "." followed by the identifier and parameters
- Methods are "pushed" onto a structure in memory called a "Call Stack"

<u>Syntax for Internal Call</u>

```
this.<<method identifier>>(<<parameters>>);
```

<u>Syntax for External Call</u>

```
<<object identifier>>.<<method identifier>>(<<parameters>>);
```

<u>Example for External</u>

```
public static void main(String [] args)
{
  AClass aClass = new aClass();
  aClass.callPublicMethod();//External call
}
```

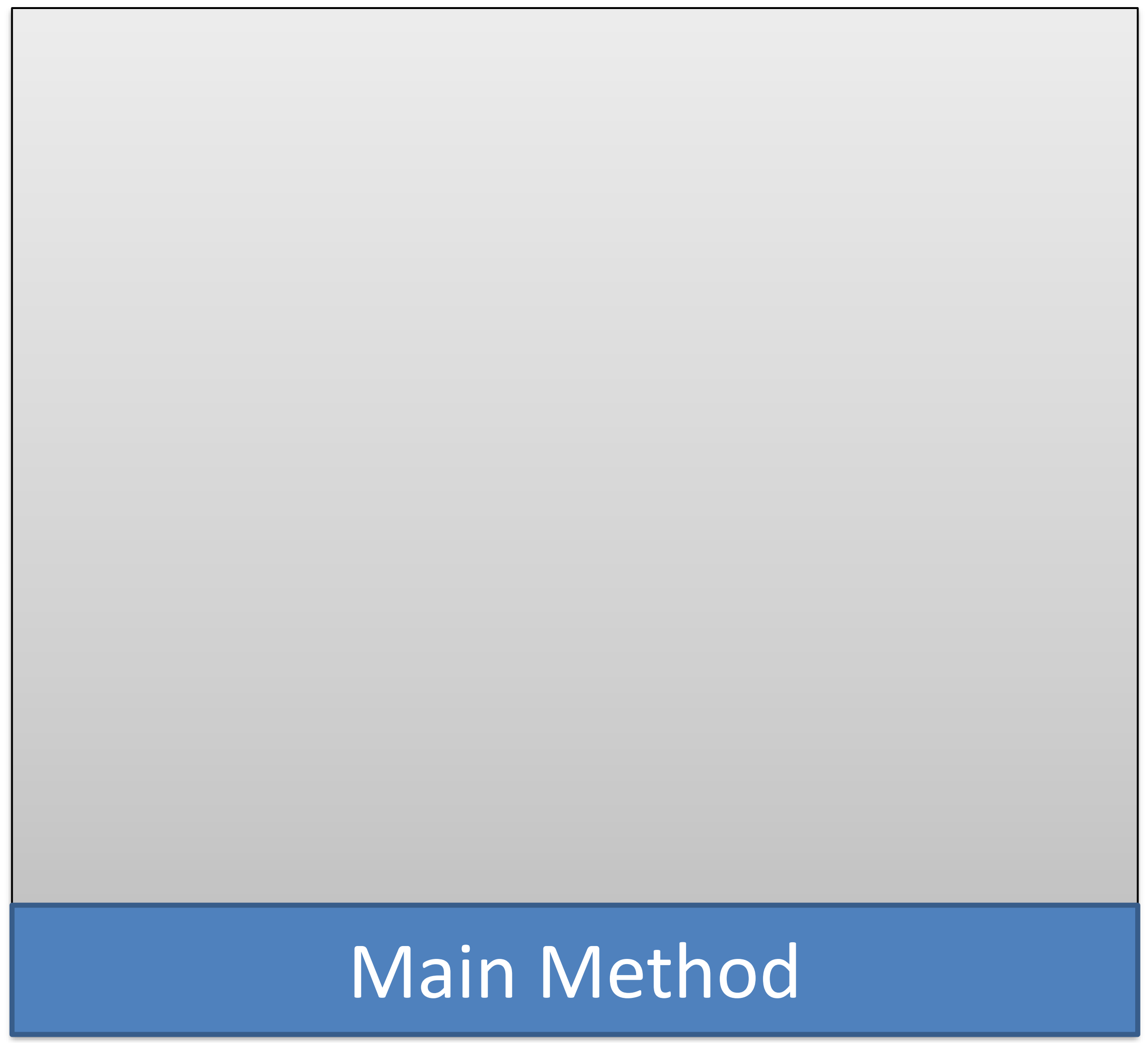
File Edit View Help

Calling Methods

Example

```
//Assume this is called from the Main
//Method
public int getValueFrom(int index)
{
  if(!isValid(index))
    return -1;
  else
    return a[index];
}
private boolean isValid(int i)
{
  return i >= 0 && i <a.length;
}
```

Call Stack in Memory

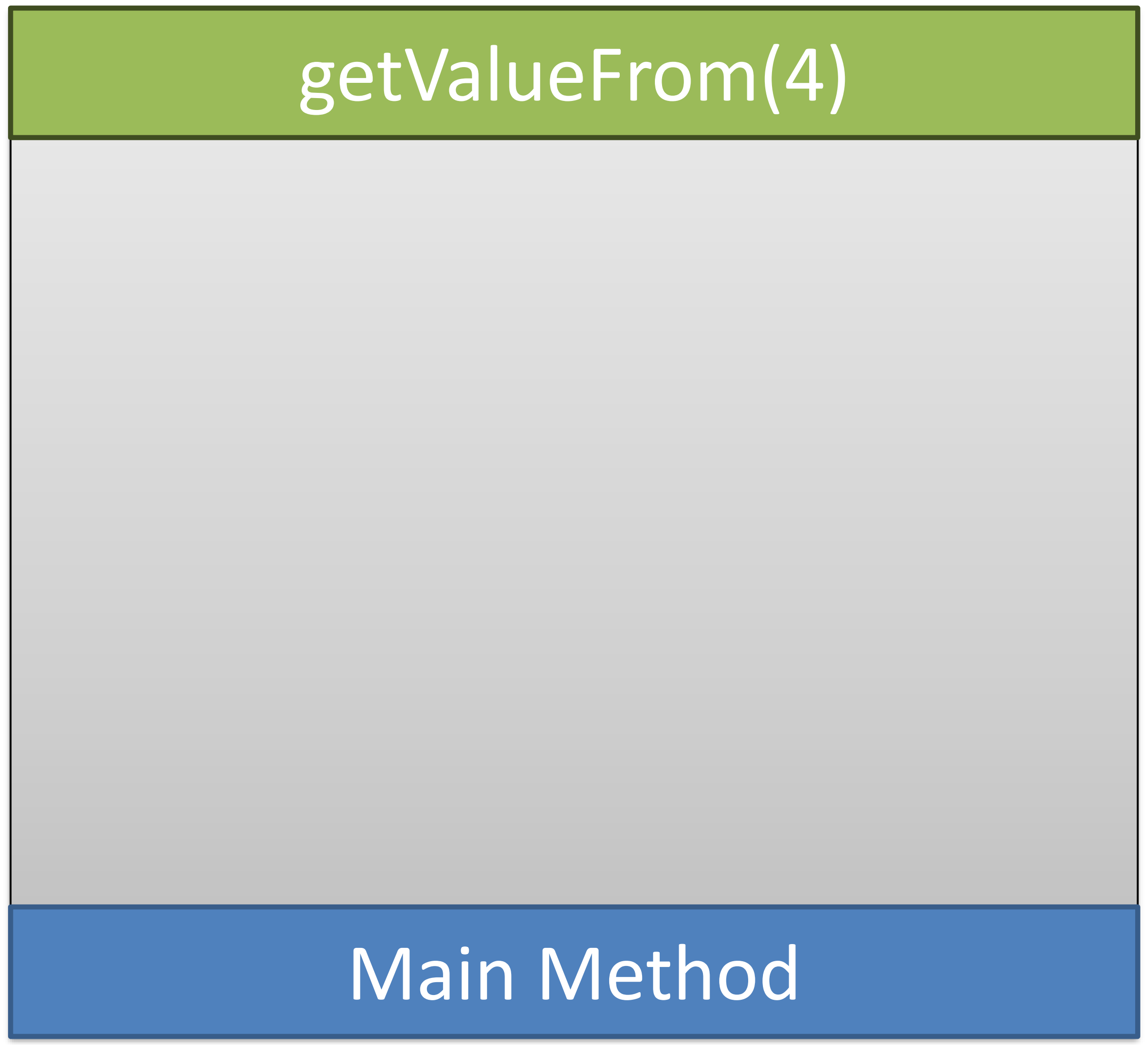


Calling Methods

Example

```
//Assume this is called from the Main
//Method
public int getValueFrom(int index)
{
  if(!isValid(index))
    return -1;
  else
    return a[index];
}
private boolean isValid(int i)
{
  return i >= 0 && i <a.length;
}
```

Call Stack in Memory



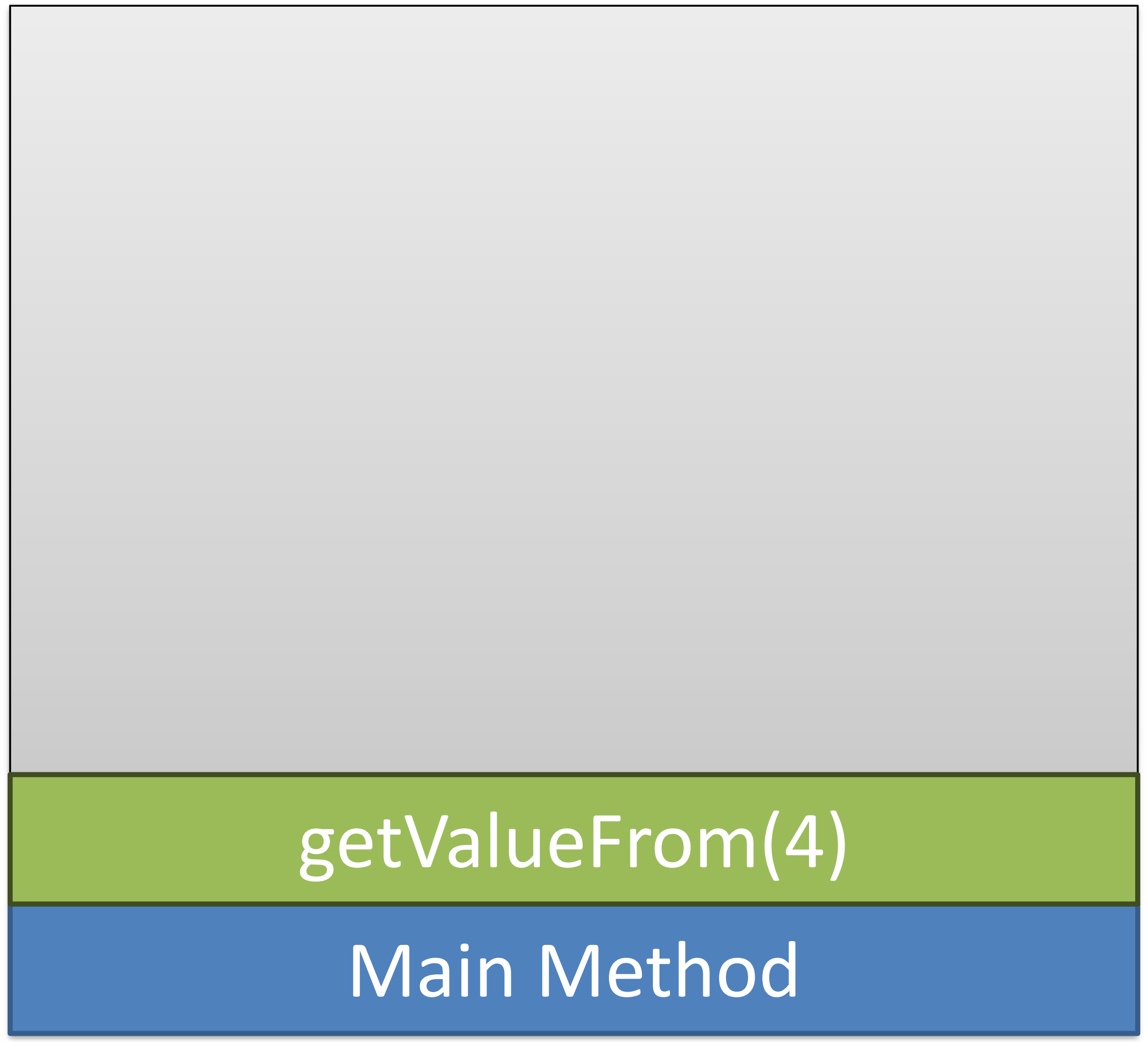
File Edit View Help

Calling Methods

Example

```
//Assume this is called from the Main
//Method
public int getValueFrom(int index)
{
  if(!isValid(index))
    return -1;
  else
    return a[index];
}
private boolean isValid(int i)
{
  return i >= 0 && i <a.length;
}
```

Call Stack in Memory

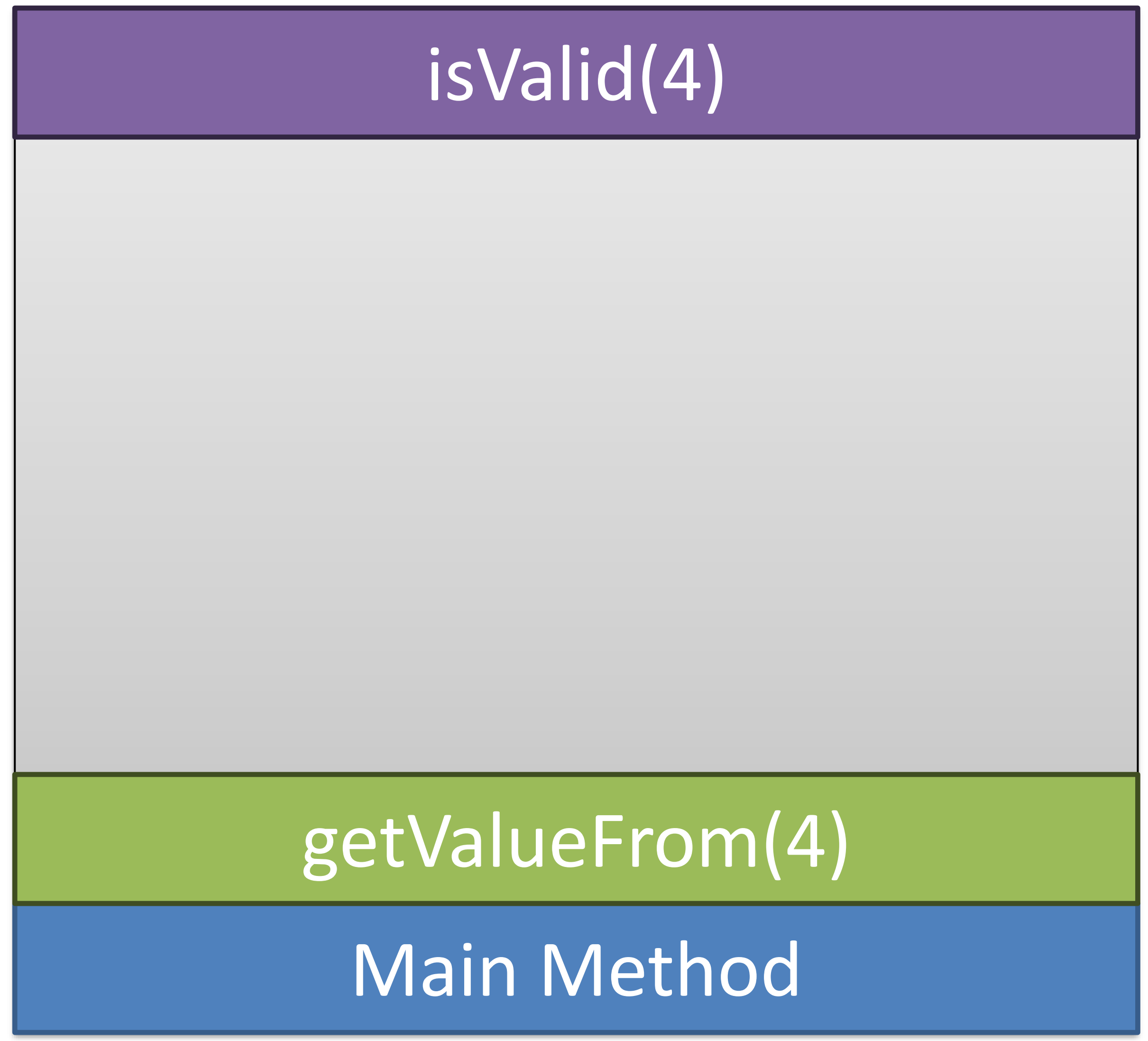


Calling Methods

Example

```
//Assume this is called from the Main
//Method
public int getValueFrom(int index)
{
  if(!isValid(index))
    return -1;
  else
    return a[index];
}
private boolean isValid(int i)
{
  return i >= 0 && i <a.length;
}
```

Call Stack in Memory



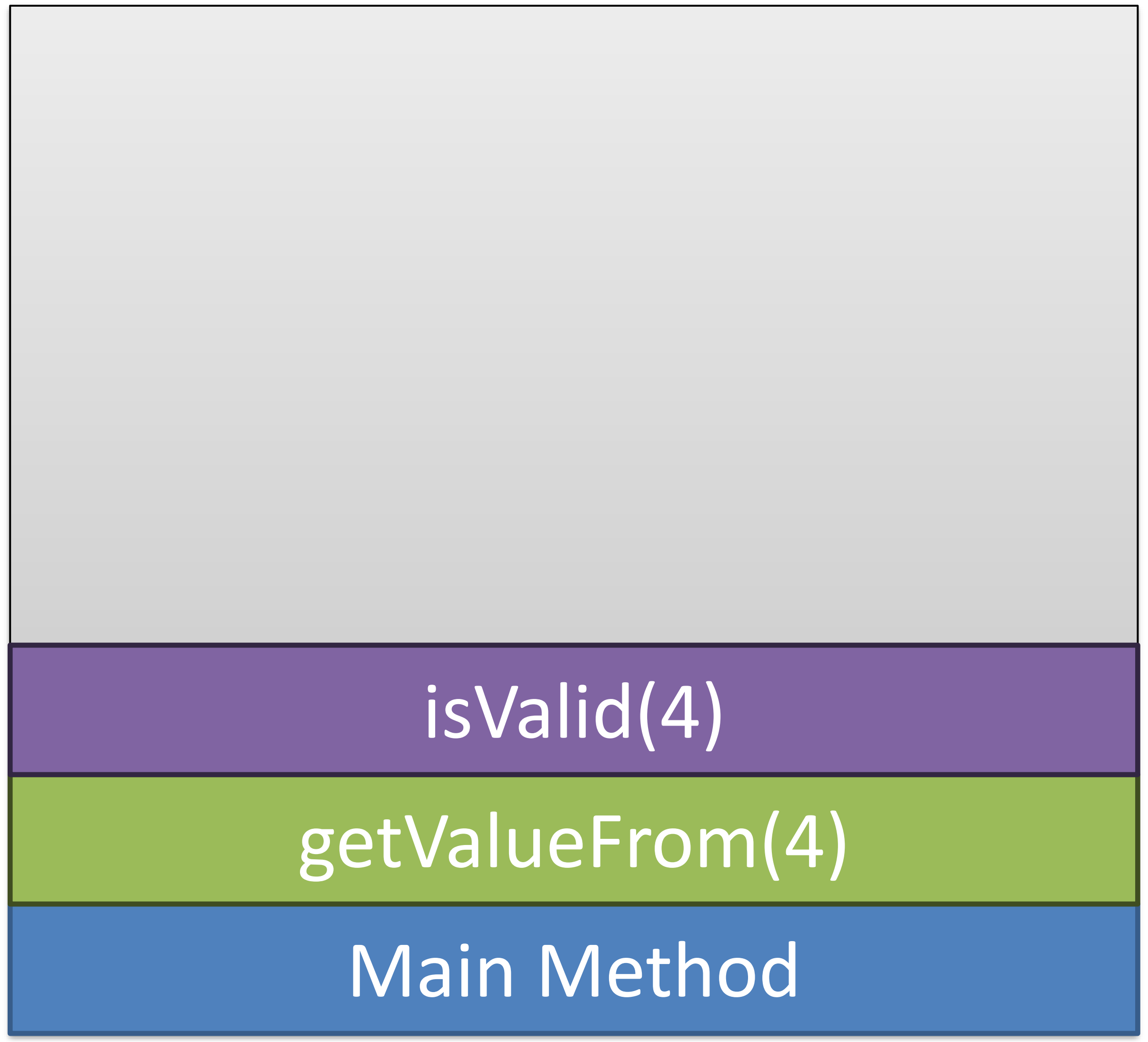
Calling Methods

Example

```
//Assume this is called from the Main
//Method
public int getValueFrom(int index)
{
  if(!isValid(index))
    return -1;
  else
    return a[index];
}
private boolean isValid(int i)
{
  return i >= 0 && i <a.length;
}
```

True

Call Stack in Memory



File Edit View Help

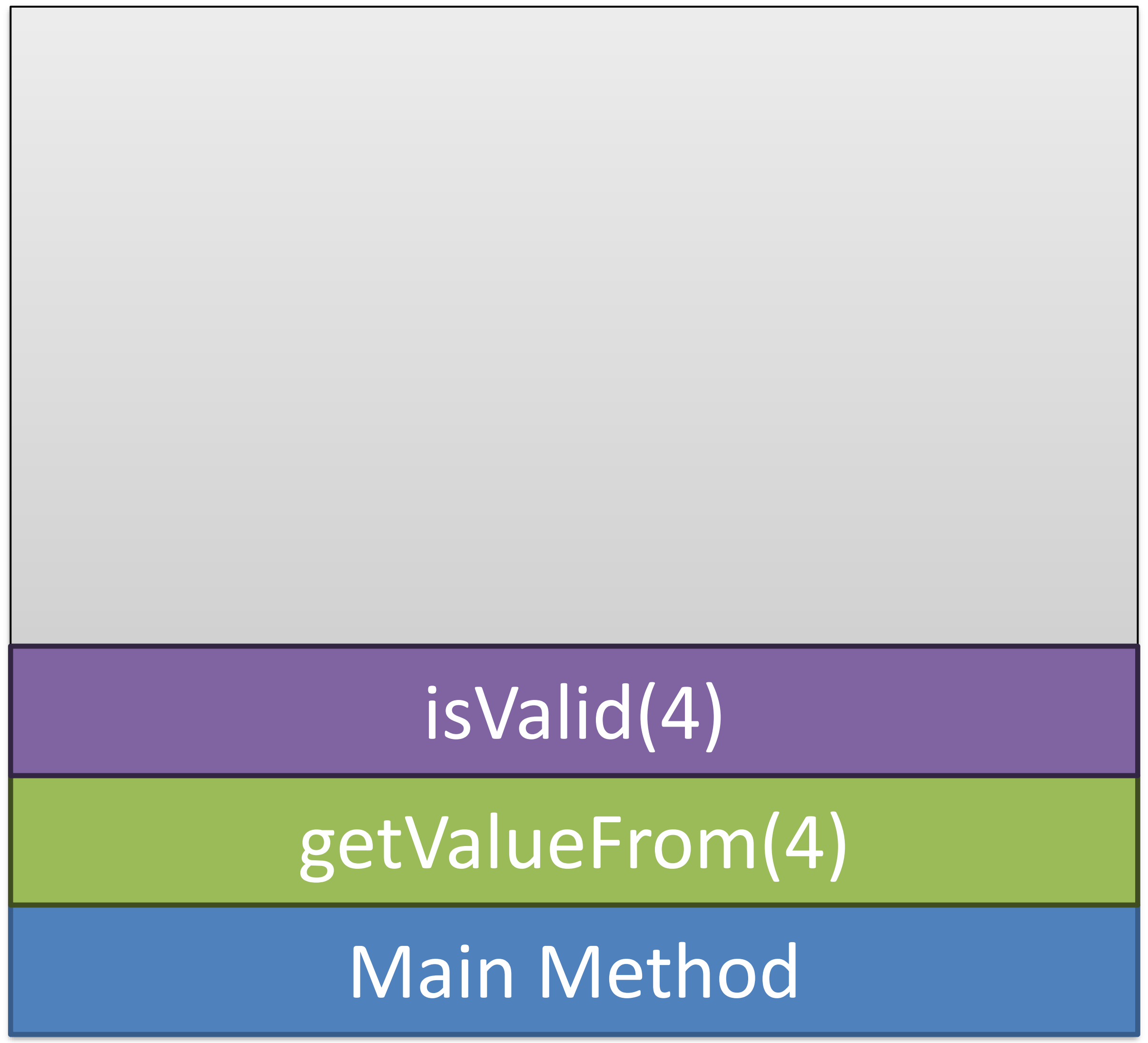
Calling Methods

Example

```
//Assume this is called from the Main
//Method
public int getValueFrom(int index)
{
  if(!isValid(index))
    return -1;
  else
    return a[index];
}
private boolean isValid(int i)
{
  return i >= 0 && i <a.length;
}
```

True

Call Stack in Memory



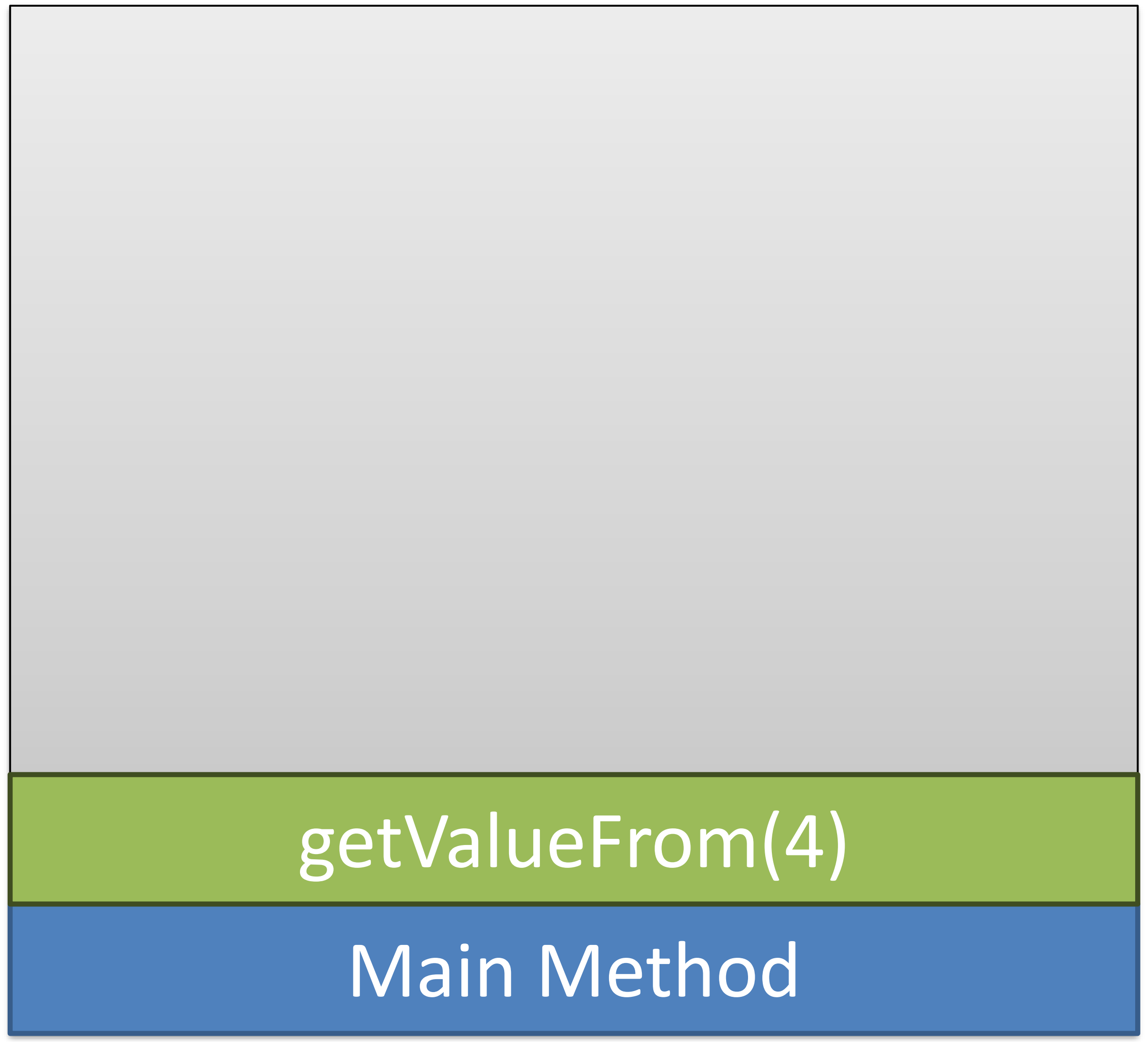
File Edit View Help

Calling Methods

Example

```
//Assume this is called from the Main
//Method
public int getValueFrom(int index)
{        !True == False
  if(!isValid(index))
    return -1;
  else
    return a[index];
}
private boolean isValid(int i)
{
  return i >= 0 && i <a.length;
}
```

Call Stack in Memory



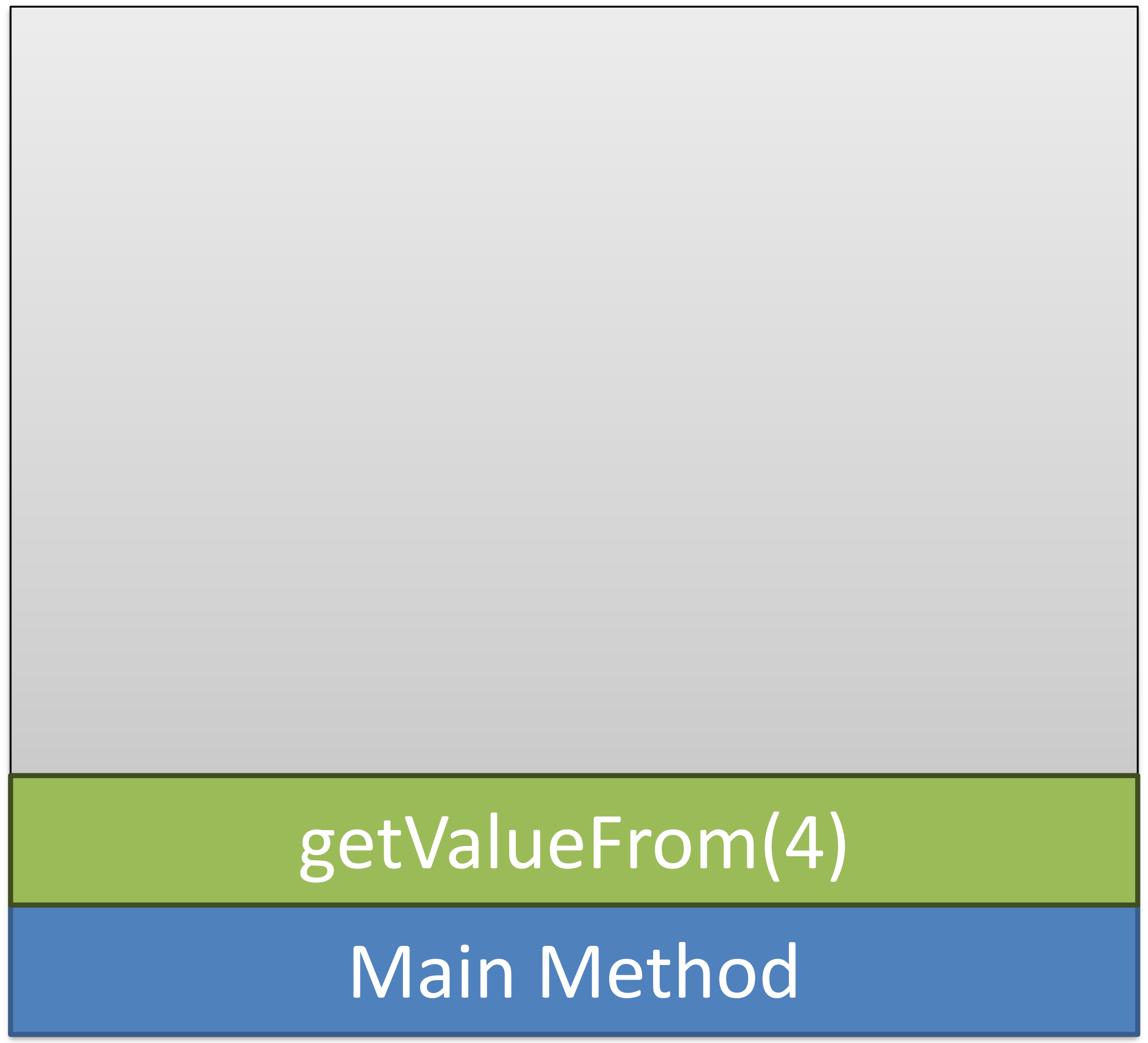
File Edit View Help

Calling Methods

Example

```
//Assume this is called from the Main
//Method
public int getValueFrom(int index)
{       !True == False
  if(!isValid(index))
    return -1;
  else
    return a[index];
}
private boolean isValid(int i)
{
  return i >= 0 && i <a.length;
}
```

Call Stack in Memory



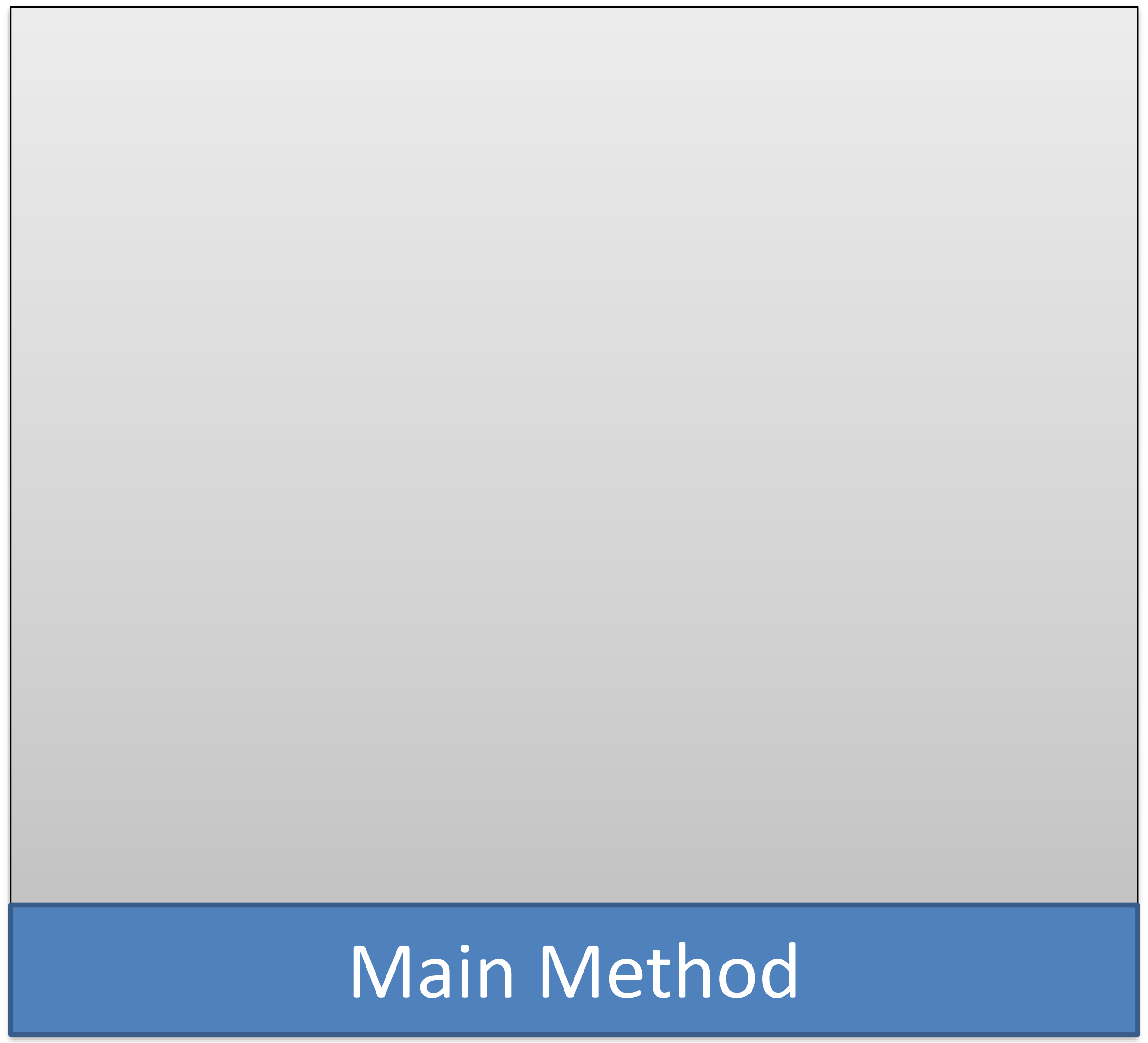
File Edit View Help

Calling Methods

Example

```
//Assume this is called from the Main
//Method
public int getValueFrom(int index)
{
  if(!isValid(index))
    return -1;
  else
    return a[index];
}
private boolean isValid(int i)
{
  return i >= 0 && i <a.length;
}
```

Call Stack in Memory



Static Methods

Syntax for Declaring Static Methods

```
<<scope>> static <<return type>> <<identifier>> (<<parameter(s)>>, …)
{
    <<Body of the Method>>
}
```

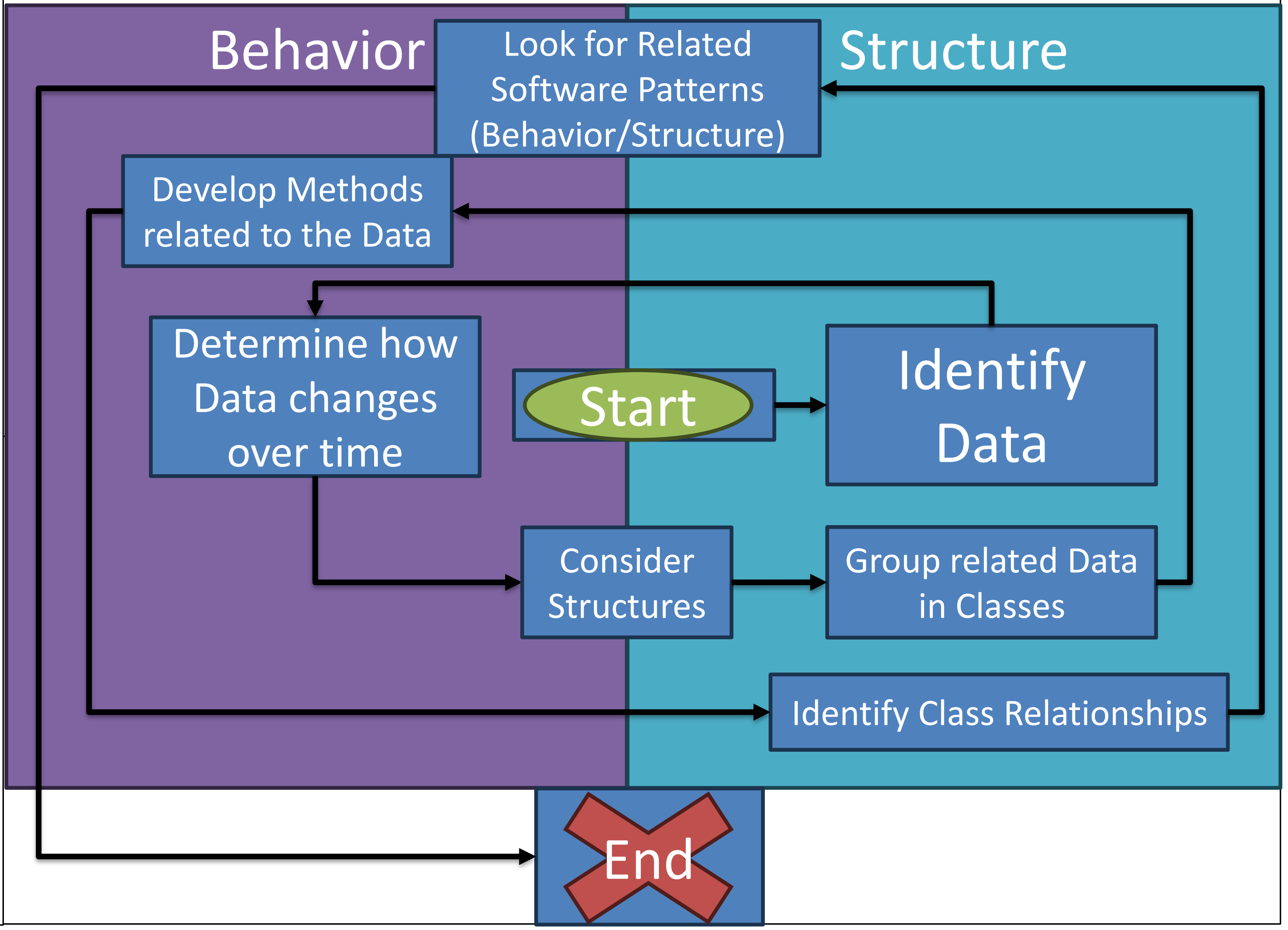
- Statically created in memory at Compilation Time.
  - Does not depend on an instance of an object
  - Sometimes called "Class Methods"
- The reserved word "this" cannot be used
- Static methods can call static methods
  - Main method can directly call other static methods
- Dynamic methods CAN call Static methods Directly
- Static methods CANNOT call Dynamic methods Directly

Example

```
public static void printError(String msg)
{
  System.out.println("Error! "+msg);
}
```

Problem Solving

4. Group together (encapsulate) related information into *Classes of Objects*.
   – How is information related and is it possible to group information in its own unique data-type?
   – In Object Orient Languages, **Classes** a structures where data and functionality can be grouped together to create instances we call **Objects.**
   – Object Oriented Programming (OOP)
   – Encapsulation
5. Develop functionality / methodologies that relates to the behavior of your Objects.
   – Objects can be viewed as *nouns* that perform actions or *verbs*.
   – Functions or methods are typically grouped inside of classes to perform actions related to the object's data.



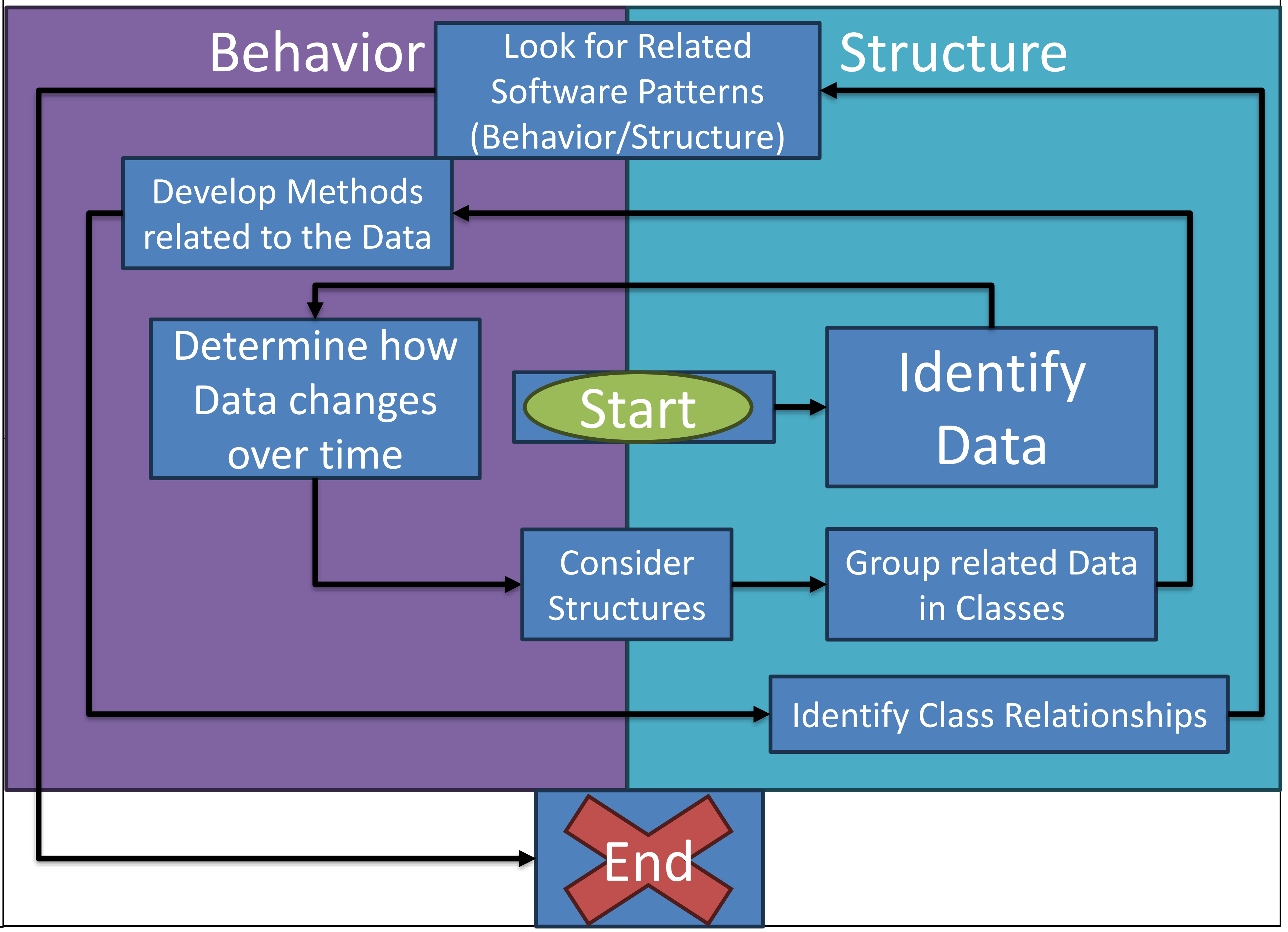
Behavior
Structure
Look for Related Software Patterns (Behavior/Structure)
Develop Methods related to the Data
Determine how Data changes over time
Start
Identify Data
Consider Structures
Group related Data in Classes
Identify Class Relationships
End

File Edit View Help

Classes and Objects

<u>7 Steps for Creating and Using a Class</u>

1. Declare the Class
2. Declare the Data
   - Instance Variables (make their scope “private”)
   - Class Constants (make their scope “public” and “static”)
3. Constructors
   - Default
   - Parameterized
4. Accessors for Every Instance Variable
5. Mutators for Every Instance Variable
   - Check for valid values
6. Other useful methods
   - toString()
   - equals(value)
7. Use it!


Behavior
Structure
Look for Related Software Patterns (Behavior/Structure)
Develop Methods related to the Data
Determine how Data changes over time
Start
Identify Data
Consider Structures
Group related Data in Classes
Identify Class Relationships
End

File Edit View Help

Classes and Objects

- Declare the Class
- The identifier becomes a Type
- Class identifier's have the same rules as Variables and Methods
  - Good programming practice to "Camel Case" these as well, but always Uppercase the first Letter
  - Good programming practice to give method "noun-like" names
- In Java the class' name must match the file name
- The scope of a class is usually public

Syntax

```
<<scope>> class <<class identifier>>
{
    <<Body of the Class>>
}
```

Example

```
public class Person
{

}
```

Classes and Objects

- Declare the Data
- The properties or attributes of a Class of Objects.
- The “Data” part of the class
- Instance Variables describes a specific instance of that class (an object)
  - Scope should be “private”
  - Encapsulation
- Class Constants describe immutable values shared by all instances of a class.
  - Scope should be “public” and it should be “static” (and “final” to make it constant)

<u>Syntax</u>

```
//Instance Variable
private <<type>> <<identifier>>;
//Class Constant
public static final <<type>> <<identifier>>;
…
```

<u>Example</u>

```
public class Person
{
    private String name;
    private int favNumber;
    public static final int DEFAULT_NUM = 0;
}
```

File Edit View Help

Classes and Objects

- Constructor are used to dynamically "Construct" an instance of a class, called an "Object", in memory during runtime.
- Replicates all code found in a Class into memory
  - The reserved word "new" precedes a constructor
  - Dynamically allocates all properties and methods
- Special kinds of Methods
  - Does not have a return type
  - Identifier must match the Class' identifier
- Default Constructor sets all properties to valid, default values
- Parameterized Constructor sets all properties to given, valid parameter values
  - Must error check (Mutators)

<u>Syntax for Default Constructor</u>

```
public <<Class Id>>()
{
	//Body of default constructor
}
```

<u>Syntax for Parameterized Constructor</u>

```
public <<Class Id>>(<<parameter>>, …)
{
	//Body of param constructor
}
```

```
public Person()
{
   this.name = "none yet";
   this.favNumber = DEFAULT_NUM;
}
public Person(String aName, int aNum)
{
   //Call mutators
}
```

File Edit View Help

Classes and Objects

- Accessors gives access to properties outside of the instance
  - The Private Scope prevents directly accessing properties like instance variables
- Create an accessor for every instance variable
- Very formulaic
  - Method's return type matches the variable's return type
  - Method's identifier starts with "get" followed by the variables identifier
  - Return the property
  - *The reserved word "this" is optional but good programming practice*

Syntax

```
public <<return type>> get<<identifier>>()
{
    return this.<<identifier>>;
}
```

Example

```
public String getName()
{
   return this.name;
}
public int getFavoriteNumber()
{
   return this.favNumber;
}
```

Classes and Objects

- Mutators gives ability to modify (mutate) the value of an Object’s property
  - –Checks for errors
- Create a mutator for every instance variable
- Very formulaic
  - –Return type is always “void”
  - –The method’s identifier is “set” followed by the variable’s identifier
  - –Has a parameter that matches the type of the variable
  - –Sets the value of the instance variable only if the parameter is valid
- Object type parameters should verify if they exist
  - –Memory address is not null

Syntax

```
public void set<<identifier>>(<<parameter>>)
{
        if(<<parameter is a valid value>>)
                this.<<instance variable>> = <<parameter>>;
        else
                this.<<instance variable>> = <<default value>>;
}
```

Example

```
public void setName(String aName)
{
    if(aName == null)
        aName = “none yet”;
    else
        this.name = aName;
}
public void setFavoriteNumber(int aNum)
{
    this.favNumber = aNum;
}
```

Classes and Objects

- Specific actions (“verbs”) that the Class of objects can do
- Two Common Useful Methods
  - toString()
  - equals(<<value>>)
- The toString() method
  - Return a String value with all properties concatenated together
  - Useful for debugging
- The equals method
  - Verifies if the properties of one object is equal to another object’s properties
  - Use this instead of “== ” for Object types
  - “==” should only be used when checking the memory address of an object type
    - When checking if the object is “null”

Syntax

```
public String toString()
{
	return <<properties concatenated together>>;
}
public boolean equals(<<other instance (oi)>>)
{
	return <<oi>> != null &&
	this.<<instance variable>> == <<oi>>.<<accessor>> &&
	this.<<instance variable>>.equals(<<oi>>.<<accessor>>) && …
}
```

Example

```
public String toString()
{
    return “Name: ”+this.name+
    “ Favorite Number: ”+this.faveNumber;
}
public boolean equals(Person aPerson)
{
    return aPerson != null &&
    this.name.equals(aPerson.getName()) &&
    this.favNumber == aPerson.getFavoriteNumber();
}
```

File Edit View Help

Classes and Objects

- To use a Class to create an instance, called an Object, first declare it
  - The type (name of the class) followed by an identifier
  - Just like any other variable
  - Default value is “null”
  - Declaring it does not create the object, it just creates room for a reference
    - Reference (memory address) points to the contents
- To construct the instance use the reserved word “new” followed by a call to the Class’ constructor
  - This should be assigned to the declared variable
- This is the only way to create a new instance
  - The assignment operator DOES NOT clone instances

Syntax

```
//Declaring
<<class type>> <<identifier>>;
//Constructing a new instance
<<identifier>> = new <<class type’s constructor>>;
```

Example

```
Person p1;
p1 = new Person();
Person p2 = new Person(“JJ”,1729);
```

File Edit View Help

Memory and Objects

- Objects in memory are separated into 2 elements
  - Reference (memory address)
  - Contents (properties and methods)
- The identifiers for Objects ONLY contain a memory address
  - "Null" is a special memory address meaning the object has not been constructed

Example

```
Person p1;
p1 = new Person();
Person p2 = new Person("JJ",1729);
Person p3 = new Person();
boolean b = (p1 == p3);//False
boolean b2 = (p1.equals(p3));//True
p1 = p2;
p1.setName("ASDF");
String name = p2.getName();// "ASDF"
```

File Edit View Help

Memory and Objects

- The assignment operator ("=") does not create *new* instances of an object.
  - Only the word "**new**" does
  - Multiple identifiers can reference the same object (Shallow Copy)
  - Cloning Objects require a *new* object created via a constructor or a clone method (Deep Copy)
- The "==" checks the memory address for objects, but not their contents
  - Should only be used when referring to the object's memory address, such as checking for null
  - Equals method should be used to check contents
- Unreachable objects are removed in Java

<u>Example</u>

```
Person p1;
p1 = new Person();
Person p2 = new Person("JJ",1729);
Person p3 = new Person();
boolean b = (p1 == p3);//False
boolean b2 = (p1.equals(p3));//True
p1 = p2;
p1.setName("ASDF");
String name = p2.getName();// "ASDF"
```

Programming Review
Part 03