File Edit View Help

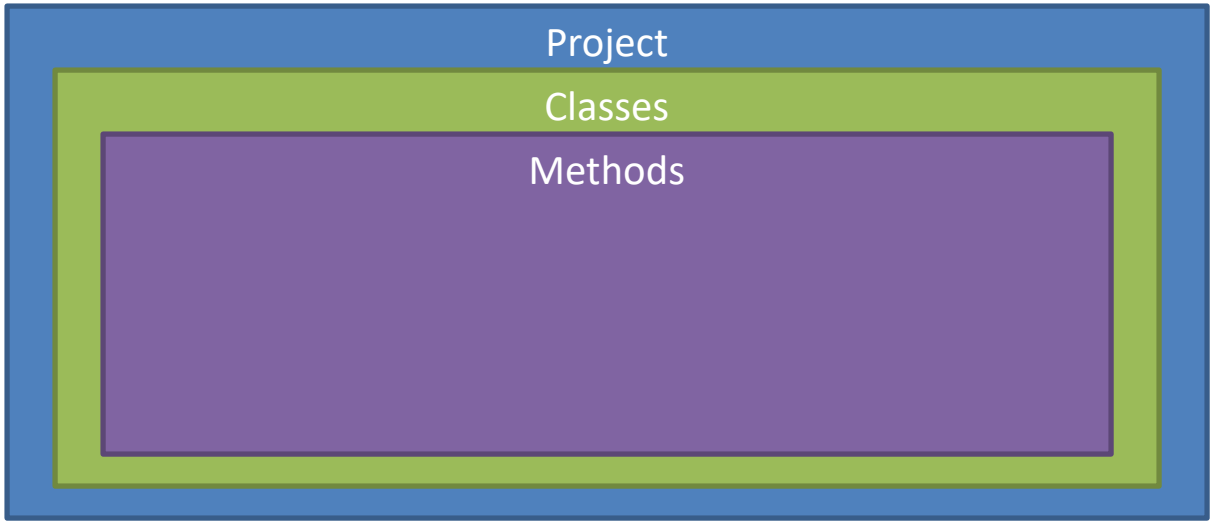
File Edit View Help

More Programming Review

File Edit View Help

Code Organization

File Edit View Help



Methods

- Groups functionality into a “callable” structure
- “Verbs”
- Create methods based on singular verbs
- Can be either Dynamic or Static
- Methods in Java must be within a Class

Syntax for Dynamic Methods

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

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 = method can be called outside of the class
  - Gives access to the functionality
- Private = method can only be called inside of the class it was defined
  - Great for creating "helper" methods that are better organize code, while preventing other programmers from using it

- 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

Example

```
public double feet2Inches(double feet)
{
      if(feet < 0.0)
            return 0.0;
      else
            return feet * 12.0;
}
```

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
- Parameters 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 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
    - 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”

Syntax for Internal

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

Syntax for External

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

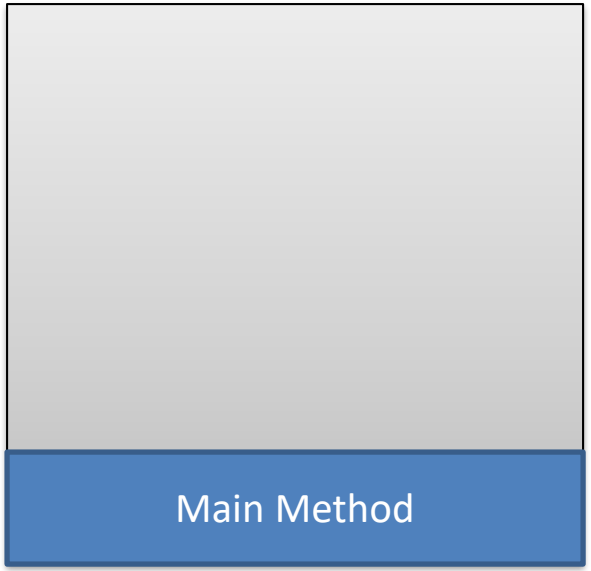
Internal Example

```
public int getValueFromArray(int index)
{
      if(!this.isValid(index))
            return -1;
      else
            return a[index];
}
```

Calling Methods

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

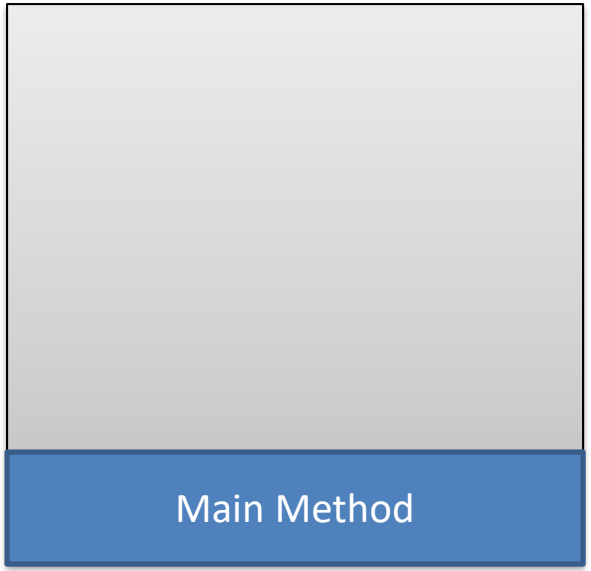
Call Stack in Memory



Calling Methods

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

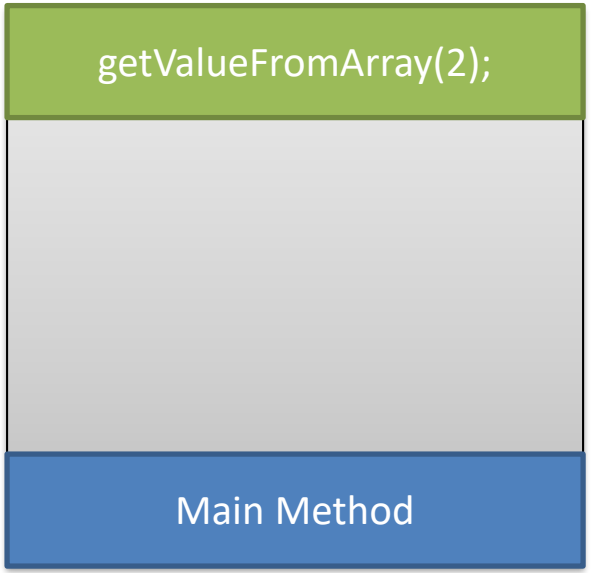
Call Stack in Memory



Calling Methods

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

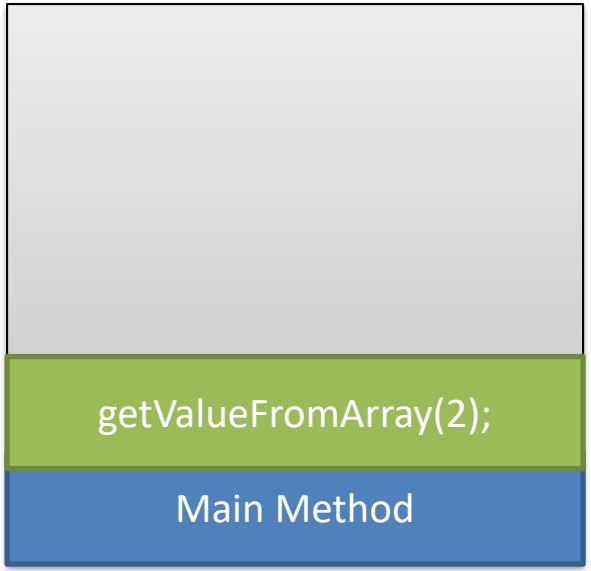
Call Stack in Memory



Calling Methods

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

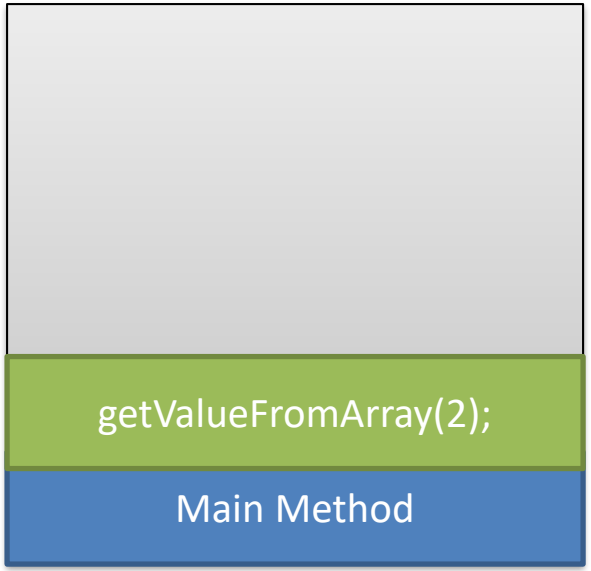
Call Stack in Memory



Calling Methods

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

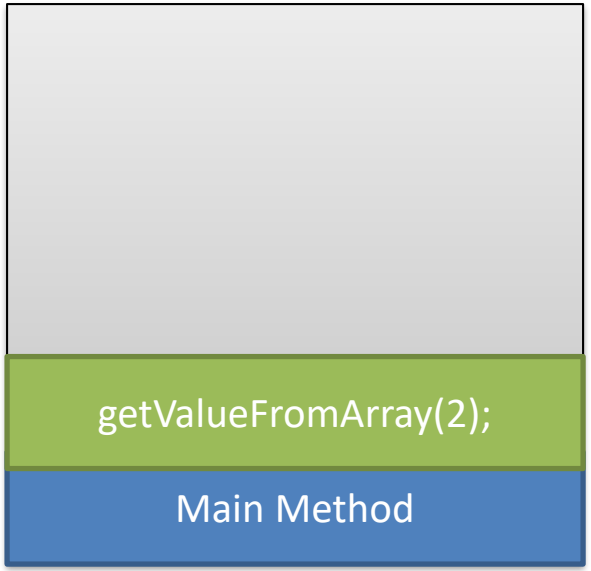
Call Stack in Memory



Calling Methods

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

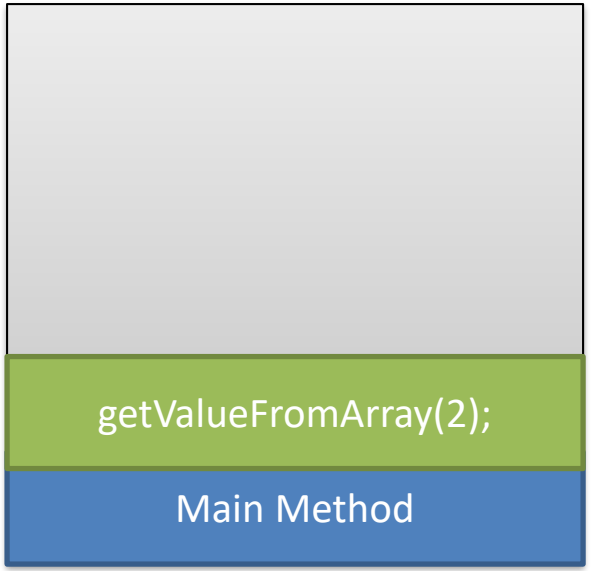
Call Stack in Memory



Calling Methods

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

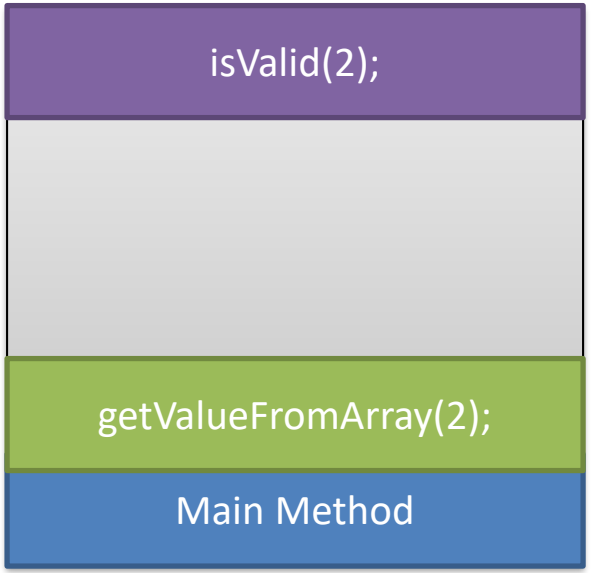
Call Stack in Memory



Calling Methods

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

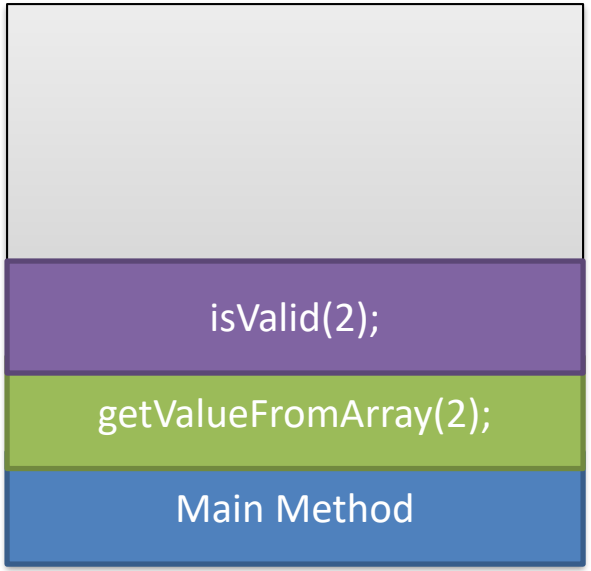
Call Stack in Memory



Calling Methods

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

Call Stack in Memory

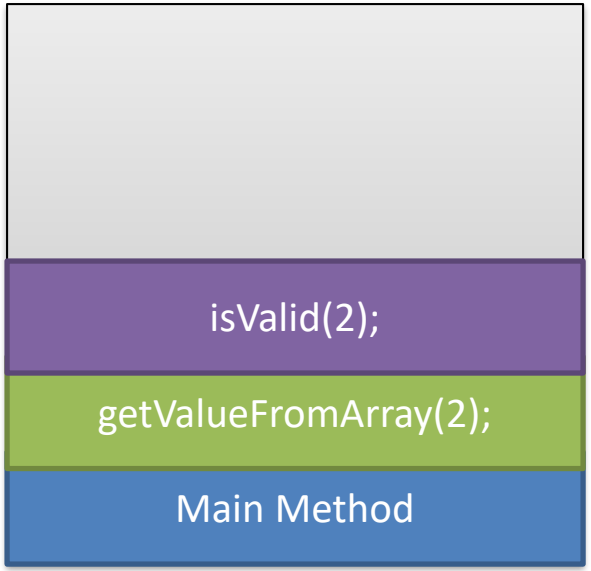


Calling Methods

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

true

Call Stack in Memory

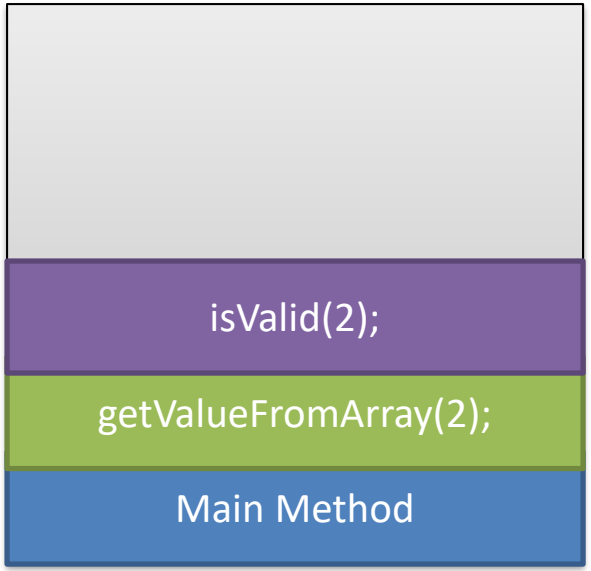


Calling Methods

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

true

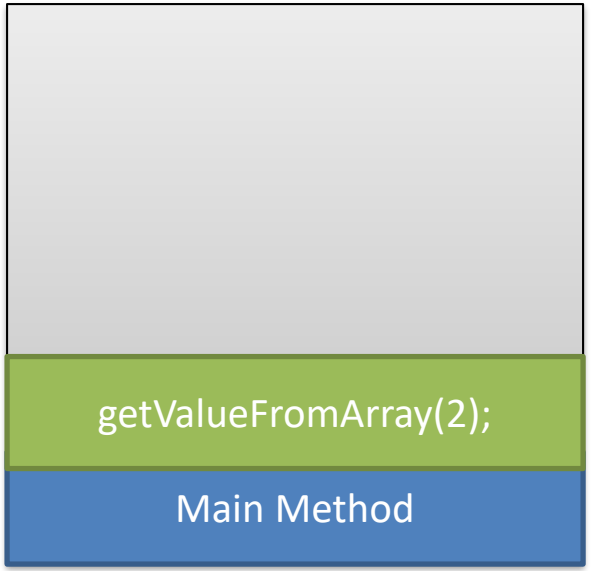
Call Stack in Memory



Calling Methods

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

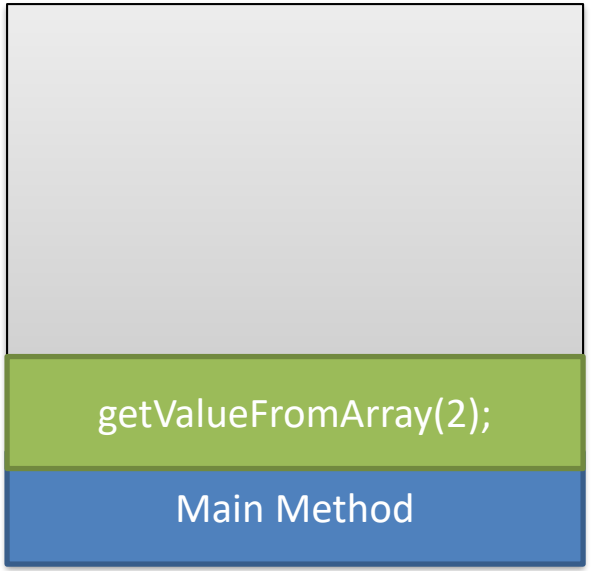
Call Stack in Memory



Calling Methods

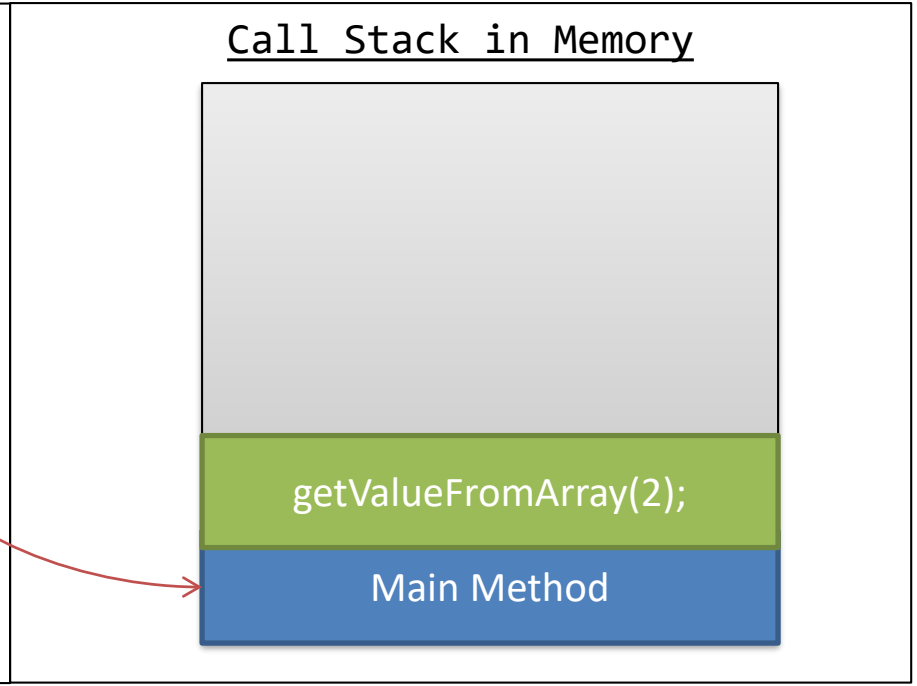
```
//Assume this is called from the Main Method
//Assume the index is valid
public int getValueFromArray(int index)
{
	if(!this.isValid(index))
		return -1;
	else
		return a[index];
}
public boolean isValid(int index)
{
	return index >= 0 && index <a.length;
}
```

Call Stack in Memory



Calling Methods

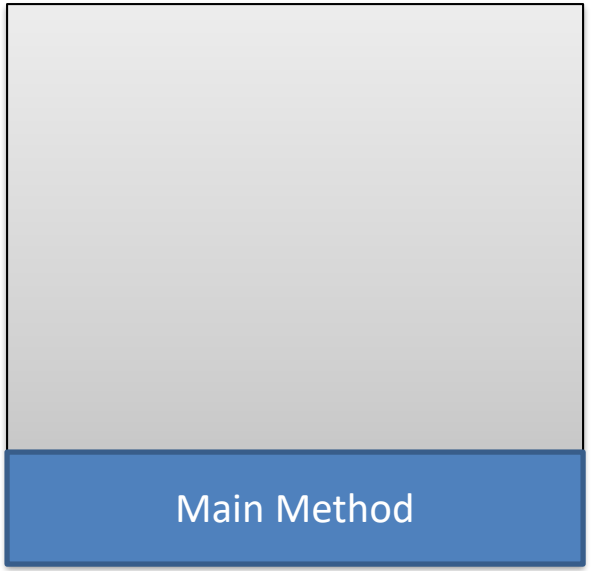
```
//Assume this is called from the Main Method
//Assume the index is valid
public int getValueFromArray(int index)
{
	if(!this.isValid(index))
		return -1;
	else
		return a[index];
}
public boolean isValid(int index)
{
	return index >= 0 && index <a.length;
}
```



Calling Methods

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

Call Stack in Memory



Static Methods

- Statically created in memory
- Created when the class is created
  - Does not depend on an instance of an object
  - Sometimes called “Class Methods”
- Great for creating functionality that does not require an instance of an class (object)
  - Shares functionality across all instances instead of a instance specific
- 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
- Static methods CANNOT call Dynamic methods

Syntax for Static Methods

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

Example

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

Object Oriented Programming (OOP)

- Use “objects” to organize code
- Key Concepts
  - Encapsulation
  - Inheritance
  - Polymorphism
- Classes
  - Group data with functionality
  - “Nouns”
  - Creates replicable code dynamically
- Objects = specific (constructed) instances of classes
- A Class becomes a Type
  - Object Type
  - Reference separated from contents
- A Class creates instances of objects

7 Steps for Creating and Using a Class

1. Define the Class
2. Declare Instance Variables
   - Make their scope “private”
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!

Define 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
{

}
```

Instance Variables

- The properties of a Class
- The “Data” part of the class
- Describes a specific instance of that class (an object)
- Scope should be “private”
  - Encapsulation

Syntax

```
private <<type>> <<identifier>>;
…
```

Example

```
public class Person
{
     private String name;
     private int favoriteNumber;
}
```

Constructors

- Used to “Construct” an instance of a class / an object in memory
- Replicates all code found in a Class into memory
  - The reserved word “new” precedes a constructor
  - Dynamically allocates all properties and methods found in a Class for that unique instance
- Special kinds of Methods
  - Does not have a return type
  - Identifier has to match the Class’ identifier
- Default Constructor sets all properties to default values
- Parameterized Constructor sets all properties to given, valid parameter values
  - Must error check
  - Calling Mutators are a good idea

Syntax for Default Constructor

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

Syntax for Parameterized Constructor

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

Example

```
public Person()
{
	this.name = “none yet”;
	this.favoriteNumber = 0;
}
public Person(String aName, int aNum)
{
	//Call mutators
}
```

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.favoriteNumber;
}
```

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.favoriteNumber = aNum;
}
```

Other Useful Methods

- 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.favoriteNumber;
}
public boolean equals(Person aPerson)
{
        return aPerson != null &&
        this.name.equals(aPerson.getName()) &&
        this.favoriteNumber == aPerson.getFavoriteNumber();
}
```

Use it!

- 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”,7);
```

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
- The assignment operator (“=“) does not create instances of an object
  - Only the word “new” does
  - Multiple identifiers can reference the same object
- 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

Example

```
Person p1;
p1 = new Person();
Person p2 = new Person(“JJ”,7);
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

Example

```
Person p1;
p1 = new Person();
Person p2 = new Person("JJ",7);
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"
```

Memory

Identifier	Contents	Byte Address
…	…	…
		28

Memory and Objects

Example

```
Person p1;
p1 = new Person();
Person p2 = new Person("JJ",7);
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"
```

Memory

Identifier	Contents	Byte Address
…	…	…
p1	NULL	28
…	…	…

Memory and Objects

Example

```
Person p1;
p1 = new Person();
Person p2 = new Person("JJ",7);
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"
```

Memory

Identifier	Contents	Byte Address
…	…	…
p1	NULL	28
…	…	…

Memory and Objects

Example

```
Person p1;
p1 = new Person();
Person p2 = new Person("JJ",7);
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"
```

Memory

Identifier	Contents	Byte Address
…	…	…
p1	NULL	28
…	…	…
p1.name	"none yet"	44
p1.favoriteNumber	0	50
…	…	…

Memory and Objects

Example

```
Person p1;
p1 = new Person();
Person p2 = new Person("JJ",7);
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"
```

Memory

Identifier	Contents	Byte Address
…	…	…
p1	44	28
…	…	…
p1.name	"none yet"	44
p1.favoriteNumber	0	50
…	…	…

Memory and Objects

Example

```
Person p1;
p1 = new Person();
Person p2 = new Person("JJ",7);
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"
```

Memory

Identifier	Contents	Byte Address
…	…	…
p1	44	28
p2	NULL	34
…	…	…
p1.name	"none yet"	44
p1.favoriteNumber	0	50
…	…	…

Memory and Objects

Example

```
Person p1;
p1 = new Person();
Person p2 = new Person("JJ",7);
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"
```

Memory

Identifier	Contents	Byte Address
…	…	…
p1	44	28
p2	NULL	34
…	…	…
p1.name	"none yet"	44
p1.favoriteNumber	0	50
…	…	…
p2.name	"JJ"	120
p2.favoriteNumber	7	126
…	…	…

Memory and Objects

Example

```
Person p1;
p1 = new Person();
Person p2 = new Person("JJ",7);
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"
```

Memory

Identifier	Contents	Byte Address
…	…	…
p1	44	28
p2	120	34
…	…	…
p1.name	"none yet"	44
p1.favoriteNumber	0	50
…	…	…
p2.name	"JJ"	120
p2.favoriteNumber	7	126
…	…	…

Memory and Objects

Example

```
Person p1;
p1 = new Person();
Person p2 = new Person("JJ",7);
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"
```

Memory

Identifier	Contents	Byte Address
…	…	…
p1	44	28
p2	120	34
p3	NULL	40
…	…	…
p1.name	"none yet"	44
p1.favoriteNumber	0	50
…	…	…
p2.name	"JJ"	120
p2.favoriteNumber	7	126
…	…	…

Memory and Objects

Example

```
Person p1;
p1 = new Person();
Person p2 = new Person("JJ",7);
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"
```

Memory

Identifier	Contents	Byte Address
…	…	…
p1	44	28
p2	120	34
p3	NULL	40
…	…	…
p1.name	"none yet"	44
p1.favoriteNumber	0	50
…	…	…
p2.name	"JJ"	120
p2.favoriteNumber	7	126
…	…	…
p3.name	"none yet"	256
p3.favoriteNumber	0	262

Memory and Objects

Example

```
Person p1;
p1 = new Person();
Person p2 = new Person("JJ",7);
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"
```

Memory

Identifier	Contents	Byte Address
…	…	…
p1	44	28
p2	120	34
p3	256	40
…	…	…
p1.name	"none yet"	44
p1.favoriteNumber	0	50
…	…	…
p2.name	"JJ"	120
p2.favoriteNumber	7	126
…	…	…
p3.name	"none yet"	256
p3.favoriteNumber	0	262

Memory and Objects

Example

```
Person p1;
p1 = new Person();
Person p2 = new Person("JJ",7);
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"
```

Memory

Identifier	Contents	Byte Address
…	…	…
p1	**44**	28
p2	120	34
p3	**256**	40
…	…	…
p1.name	"none yet"	44
p1.favoriteNumber	0	50
…	…	…
p2.name	"JJ"	120
p2.favoriteNumber	7	126
…	…	…
p3.name	"none yet"	256
p3.favoriteNumber	0	262

Memory and Objects

Example

```
Person p1;
p1 = new Person();
Person p2 = new Person("JJ",7);
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"
```

Memory

Identifier	Contents	Byte Address
…	…	…
p1	44	28
p2	120	34
p3	256	40
…	…	…
p1.name	**"none yet"**	44
p1.favoriteNumber	**0**	50
…	…	…
p2.name	"JJ"	120
p2.favoriteNumber	7	126
…	…	…
p3.name	**"none yet"**	256
p3.favoriteNumber	**0**	262

Memory and Objects

Example

```
Person p1;
p1 = new Person();
Person p2 = new Person("JJ",7);
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"
```

Memory

Identifier	Contents	Byte Address
…	…	…
p1	44	28
p2	120	34
p3	256	40
…	…	…
p1.name	"none yet"	44
p1.favoriteNumber	0	50
…	…	…
p2.name	"JJ"	120
p2.favoriteNumber	7	126
…	…	…
p3.name	"none yet"	256
p3.favoriteNumber	0	262

Memory and Objects

Example

```
Person p1;
p1 = new Person();
Person p2 = new Person("JJ",7);
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"
```

Memory

Identifier	Contents	Byte Address
…	…	…
p1	120	28
p2	120	34
p3	256	40
…	…	…
p1.name	"none yet"	44
p1.favoriteNumber	0	50
…	…	…
p2.name	"JJ"	120
p2.favoriteNumber	7	126
…	…	…
p3.name	"none yet"	256
p3.favoriteNumber	0	262

Memory and Objects

Example

```
Person p1;
p1 = new Person();
Person p2 = new Person("JJ",7);
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"
```

Memory

Identifier	Contents	Byte Address
…	…	…
p1	120	28
p2	120	34
p3	256	40
…	…	…
p1.name	"none yet"	44
p1.favoriteNumber	0	50
…	…	…
p2.name	"JJ"	120
p2.favoriteNumber	7	126
…	…	…
p3.name	"none yet"	256
p3.favoriteNumber	0	262

Memory and Objects

Example

```
Person p1;
p1 = new Person();
Person p2 = new Person("JJ",7);
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"
```

Memory

Identifier	Contents	Byte Address
…	…	…
p1	120	28
p2	120	34
p3	256	40
…	…	…
p1.name	"none yet"	44
p1.favoriteNumber	0	50
…	…	…
p2.name	"JJ"	120
p2.favoriteNumber	7	126
…	…	…
p3.name	"none yet"	256
p3.favoriteNumber	0	262

Memory and Objects

Example

```
Person p1;
p1 = new Person();
Person p2 = new Person("JJ",7);
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"
```

Memory

Identifier	Contents	Byte Address
…	…	…
p1	120	28
p2	120	34
p3	256	40
…	…	…
p1.name	"none yet"	44
p1.favoriteNumber	0	50
…	…	…
p2.name	**"ASDF"**	120
p2.favoriteNumber	7	126
…	…	…
p3.name	"none yet"	256
p3.favoriteNumber	0	262

Memory and Objects

Example

```
Person p1;
p1 = new Person();
Person p2 = new Person("JJ",7);
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"
```

Memory

Identifier	Contents	Byte Address
…	…	…
p1	120	28
p2	120	34
p3	256	40
…	…	…
p1.name	"none yet"	44
p1.favoriteNumber	0	50
…	…	…
p2.name	**"ASDF"**	120
p2.favoriteNumber	7	126
…	…	…
p3.name	"none yet"	256
p3.favoriteNumber	0	262

Memory and Objects

Example

```
Person p1;
p1 = new Person();
Person p2 = new Person("JJ",7);
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"
```

Memory

Identifier	Contents	Byte Address
…	…	…
p1	120	28
p2	120	34
p3	256	40
…	…	…
p1.name	**"none yet"**	**44**
p1.favoriteNumber	**0**	**50**
…	…	…
p2.name	"ASDF"	120
p2.favoriteNumber	7	126
…	…	…
p3.name	"none yet"	256
p3.favoriteNumber	0	262

Memory and Objects

Example

```
Person p1;
p1 = new Person();
Person p2 = new Person("JJ",7);
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"
```

Memory

Identifier	Contents	Byte Address
…	…	…
p1	120	28
p2	120	34
p3	256	40
…	…	…
~~p1.name~~	~~"none yet"~~	~~44~~
~~p1.favoriteNumber~~	~~0~~	~~50~~
…	…	…
p2.name	"ASDF"	120
p2.favoriteNumber	7	126
…	…	…
p3.name	"none yet"	256
p3.favoriteNumber	0	262

Inheritance

- Establishes a relationship between two classes where properties and methods are absorbed from one class into another
- Similar to Biological Inheritance
- Used to create a more “specific” version of a given class
  - Creates an “is a” relationship
- Reserved word “extends” is used to establish this in the Class definition
- The reserved word “super” can be used to access methods and constructors from the parent class
  - super() is used to call the parent’s constructors
  - super.<<method>> is used to call the parent’s method

Syntax

```
//Class def
public class <<class identifier>> extends <<other class>>
{
        …
        public <<class identifier>>()
        {
                super();//Call other class’ default constructor
        }
}
```

Example

```
public class Employee extends Person
{
        private int id;
        public Employee()
        {
                super();//Call to Person’s def constr
                this.id = 0;
        }
        …
        public boolean equals(Employee e)
        {
                return e != null && super.equals(e) && this.id == e.getID();
        }
}
```

Polymorphism

- "One becomes many"
- One type's methods could be implemented in several ways
- Interfaces
  - Non-constructible type
  - Only method definitions
  - Provides "rules" for other programmers
  - Provides a "variable" for classes that implement the interface
  - Interface identifier must match the filename
- Classes "implement" interfaces
  - Unlike inheritance, a class may implement any number of interfaces
  - All methods in the interface MUST BE implemented or else there will be a syntax error
- Interfaces may "extend" other interfaces

Syntax

```
//Interface def
public interface <<interface identifier>>
{
        <<method definition>>;//Semi-colon after each method def
}
//Class implementing the interface
public class <<class identifier>> implements <<interface>>
{…
```

Example

```
public interface DrawableObject
{
        public void draw();
}
…
public class Rectangle implements DrawableObject
{
        …
        public void draw()
        {
                …
        }
}
```

File Edit View Help

Code Organization