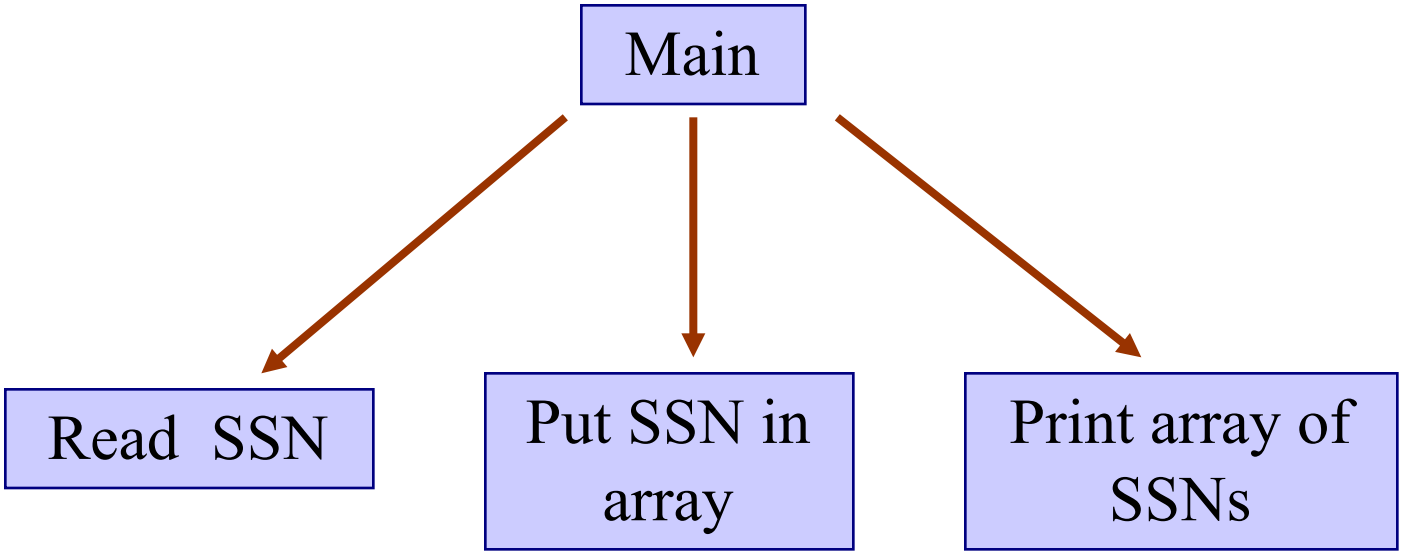
Program Modularity and Error Handling

Program Modularity

- Break a program down into smaller parts (methods).
- Test each method separately.
- Understand the relationship between the methods: parameters and their expected values

Example problem:

Read social security numbers, store them in an array, then print the array.



We can already write the program…

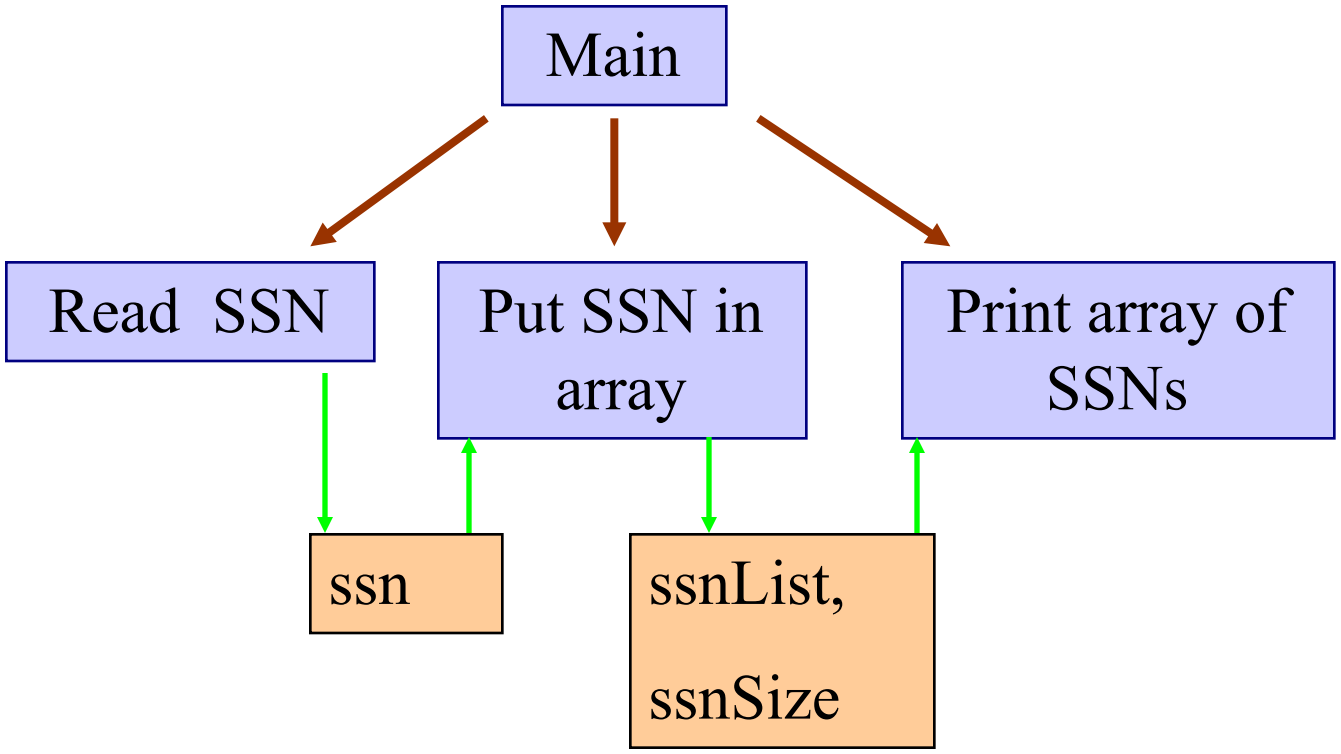
```
public class SSN {
   public static void main(String[] args) {
      readSSN();
      storeSSN();
      printSSNList();
   }
   public static String readSSN(){}
   public static void storeSSN(){}
   public static void printSSNList() {}
} //SSN
```

What about data flow?

Obvious variables:

```
String ssn;

String[] ssnList; int ssnSize;
```



Parameters represent data flow

```
public class SSN {
   static String ssn;
   static String[] ssnList;
   public static void main(String[] args) {
      ssn = readSSN();
      storeSSN(ssn,ssnList,ssnSize);
      printSSNList(ssnList,ssnSize);
   }
   public static String readSSN(){}
   public static void storeSSN(String s, String[] list, int size){}
   public static void printSSNList(String[] list, int size) {}
} //SSN
```

Simple solution… and it doesn't work!

```
static String ssn;
static String[] ssnList;
static int ssnSize;
public static void main(String[] args) {
   ssn = readSSN();
   storeSSN(ssn,ssnList,ssnSize);
   printSSNList(ssnList,ssnSize);
}
public static String readSSN(){
   return(JOptionPane.showInputDialog(null,"Enter SSN:"));
}
public static void storeSSN(String s, String[] list, int size){
   list[size++]=s;
}
public static void printSSNList(String[] list, int size) {
         for (int i=0;i<size;i++)
                  System.out.println(list[i]);
}
```

Simple solution… and it doesn't work!

```
static String ssn;
static String[] ssnList;
static int ssnSize;
public static void main(String[] args) {
   ssn = readSSN();
   storeSSN(ssn,ssnList,ssnSize);
   printSSNList(ssnList,ssnSize);
}
public static String readSSN(){
   return(JOptionPane.showInputDialog(null,"Enter SSN:"));
}
public static void storeSSN(String s, String[] list, int size){
   list[size++]=s;
}
public static void printSSNList(String[] list, int size) {
         for (int i=0;i<size;i++)
                  System.out.println(list[i]);
}
```

```
static String ssn;
static String[] ssnList;
static int ssnSize;
public static void main(String[] args) {
   ssn = readSSN();
   storeSSN(ssn,ssnList,ssnSize);
   printSSNList(ssnList,ssnSize);
}
public static String readSSN(){
   return(JOptionPane.showInputDialog(null,"Enter SSN:"));
}
public static void storeSSN(String s, String[] list){
   list[ssnSize++]=s;
}
public static void printSSNList(String[] list, int size) {
         for (int i=0;i<size;i++)
                  System.out.println(list[i]);
}
```

What about data validation?

- Data entered by a user (at a prompt)
- Data entered by a user (into a file)
- Data values received from other *methods*
- A method should verify what it receives
- A method should verify what it returns

readSSN

```
public static String readSSN(){
  return(JOptionPane.showInputDialog(null,"Enter SSN:"));
   }
```

- Don't just return the SSN entered, check it!
- What makes it valid?
- A string of exactly nine digits
- What do we do if it's not valid?? Exit program?

```
public static String readSSN(){
  String ssn;
  ssn = (JOptionPane.showInputDialog(null,"Enter SSN:"));
    if (ssn.length() != 9) {
      System.out.println("An SSN length must be 9");
      System.exit(0);
    }
    for (int i=0;i<9;i++)
      if (! Character.isDigit(ssn.charAt(i))) {
      System.out.println("SSN must have only digits.");
      System.exit(0);
    }
    return ssn;
```

storeSSN

```
public static void storeSSN(String s, String[] list){
  list[ssnSize++]=s;
}
```

- Should we assume the SSN is valid?
- Is the array full?
- Is the array valid? (It really is an object)

```
public static void storeSSN(String s, String[] list){
  if (ssn.length() != 9) {
    System.out.println("An SSN length must be 9");
    System.exit(0);
  }
  for (int i=0;i<9;i++)
    if (! Character.isDigit(ssn.charAt(i))) {
      System.out.println("SSN must have only digits.");
      System.exit(0);
    }
    if (list == null){
      System.out.println("Array is null.");
      System.exit(0);
    if (list != null && ssnSize == list.length)
      System.exit(0);
    list[ssnSize++]=s;
}
```

```
public static void storeSSN(String s, String[] list){
  if (ssn.length() != 9) {
    System.out.println("An SSN length must be 9");
    System.exit(0);
  }
  for (int i=0;i<9;i++)
    if (! Character.isDigit(ssn.charAt(i))) {
      System.out.println("SSN must have only digits.");
      System.exit(0);
    }
    if (list == null){
      System.out.println("Array is null.");
      System.exit(0);
    if (ssnSize == list.length)
      System.exit(0);
    list[ssnSize++]=s;
}
```

Wait! We just did this in *readSSN*!

```
public static void storeSSN(String s, String[] list){
  if (!isValidSSN(s))
    System.exit(0);
  if (ssnSize == list.length)
    System.exit(0);
  list[ssnSize++]=s;
}
```

```
public static String readSSN(){
  String ssn;
  ssn = (JOptionPane.showInputDialog(null,"Enter SSN:"));
    if (isValidSSN(ssn))
      return ssn;
    else
      return null;
}
```

One method for SSN validity

```
public static boolean isValidSSN(String s) {
  if (s.length() != 9) {
    System.out.println("An SSN length must be 9");
    return(false);
  }
  for (int i=0;i<9;i++)
    if (! Character.isDigit(s.charAt(i))) {
      System.out.println("SSN must have only digits.");
      return(false);
    }
  return (true);
}
```

And why not one method for list validity?

```
public static boolean isValidList(String[] list){
  if (list == null){
    System.out.println("Array is null.");
    return (false);
  }
  if (ssnSize == list.length){
    System.out.println("Can't store any more SSNs");
    return (false);
  }
  return (true);
}
```

Both methods are simpler, clearer and check errors

```
public static String readSSN(){
  String ssn;
  ssn = (JOptionPane.showInputDialog(null,"Enter SSN:"));
    if (isValidSSN(ssn))
      return ssn;
    else
    return null;
}
public static void storeSSN(String s, String[] list){
  if (isValidSSN(s) && isValidList(list))
    list[ssnSize++]=s;
}
```

Fix up *printSSNList*

```
public static void printSSNList(String[] list, int size) {
  if (!isValidList(list)){
    System.out.println("Can't print from invalid list.");
    System.exit(0);
  }
  for (int i=0;i<size;i++)
    if (!isValidSSN(list[i]))
      System.out.println("Invalid SSN: "+list[i]);
    else
      System.out.println(list[i]);
}
```

Finally, how can we test the program?

```
public static void main(String[] args) {
     initialize();
     do {
       ssn = readSSN();
       storeSSN(ssn,ssnList);
       printSSNList(ssnList,ssnSize);
     }
     while (!ssn.equals("000000000"));
  }
```

This requires typing data values one at a time into an input dialog.

Creating a file with all sorts of test cases is better.

```
public static String readSSN(){
      String ssn;
      ssn = inFile.readLine();
      if(ssn == null)
         return "000000000";
      else
         if (isValidSSN(ssn))
            return ssn;
         else
             return null;
  }
```

More about handling errors

- "Run-time" error messages can provide useful information to the programmer or to the user.
- Errors could be due to:
- Bad code (this is why we must test!)
- Bad data (can the program still continue?)

Bad code and testing

- Test the program will all kinds of possible data.
- From method to method, all data variables in the program should be in a "correct" state.
- For example, a method that sorts should produce a sorted object.

Assertions

- Assertions are used during program development, testing for errors in the logic of the program.
- They are usually "turned off" when a program is run by the user.
- Such errors should not occur in the final version of the program.

```
public static void storeSSN(String s, String[] list){
  if (!isValidSSN(s))
    System.exit(0);
  if (ssnSize == list.length)
    System.exit(0);
  list[ssnSize++]=s;
}
```

```
public static void storeSSN(String s, String[] list){
   assert (isValidSSN(s)): "The SSN is not valid";
   assert (isValidList(list)): "The array is not valid";
   if (isValidSSN(s) && isValidList(list))
     list[ssnSize++]=s;
   assert (isValidList(list)):"Resulting list not valid";
}
```

Assertions

- Once an assertion is "thrown," the program terminates.
- These error should not happen in "real life."
- The string in the assertion statement is printed along with a stack trace.

Exceptions

- An exception is an error that can be "thrown" by a method.
- For example, Integer.parseInt("abc") will throw an exception called `IllegalArgumentException`
- Our program can also throw these common exceptions (and terminate).
- Later we'll see that the program may be able to "catch" these exceptions and continue

How to *throw* an exception

IllegalArgumentException is a kind of exception. To *throw* one:

```
String message = "An SSN length must be 9";
IllegalArgumentException iae;
iae = new IllegalArgumentException(message);
throw iae;
```

or just do:

```
throw new IllegalArgumentException
                        ("An SSN length must be 9");
```

```
public static boolean isValidSSN(String s) {
  if (s.length() != 9)
    throw new IllegalArgumentException
                              ("An SSN length must be 9");
  for (int i=0;i<9;i++)
    if (! Character.isDigit(s.charAt(i)))
      throw new IllegalArgumentException
                              ("SSN must have only digits.");
       }
  return (true);
}
```