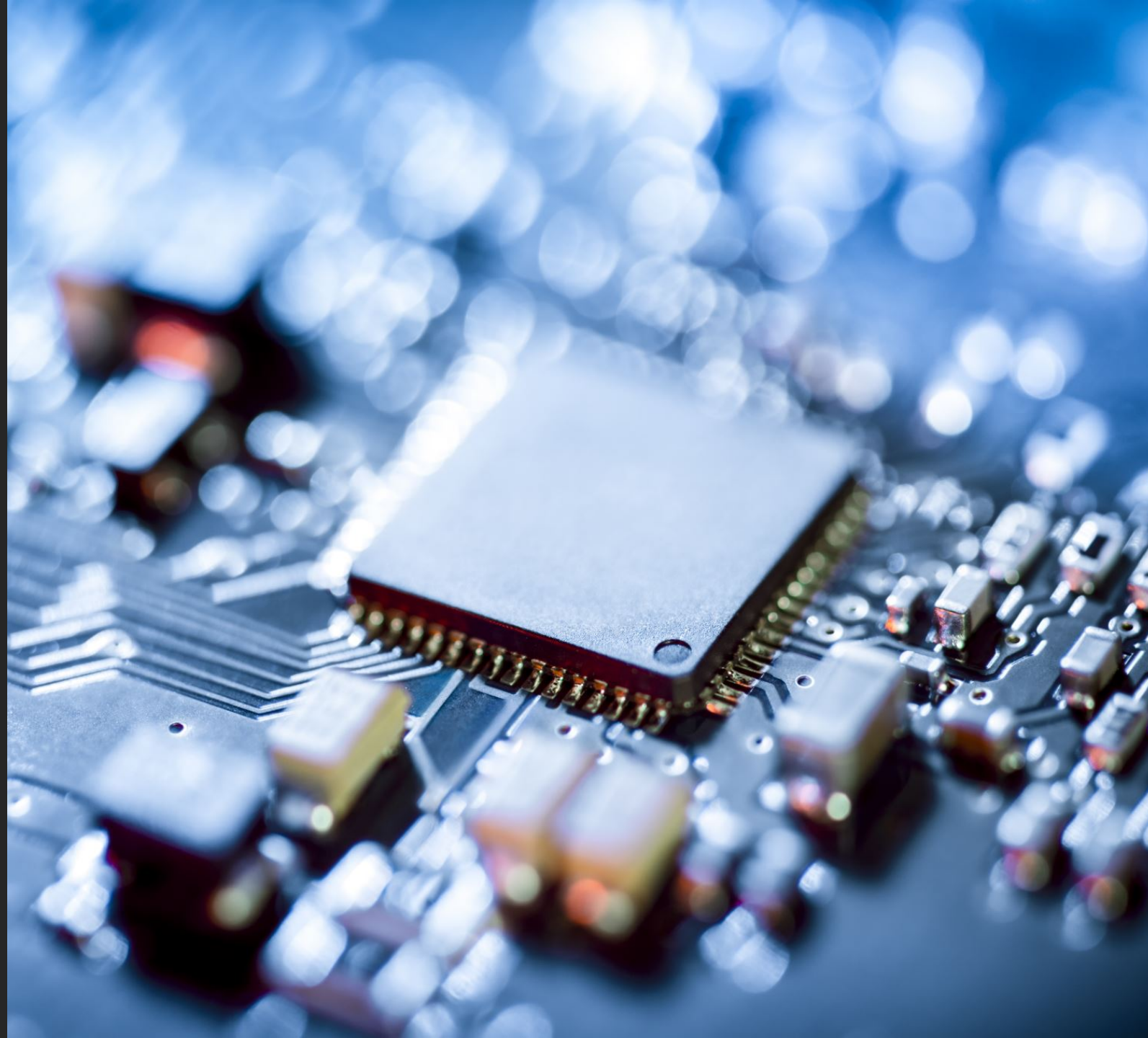
# SOFTWARE ENGINEERING

ECE 651

SPRING 2021

OO REVIEW AND BASIC IDEAS



# OBJECT ORIENTATION + GOOD DESIGN

- Review of OO Basics (551)
  - Classes, Methods, Fields
  - Has-A vs Is-A
  - Encapsulation
  - Static
- Decoupling
  - Exceptions: decouple error handling from error-producing code
- Differ in data vs in behavior

# OO REVIEW: CLASSES, METHODS, FIELDS

- A class describes a **type** of object we might create
  - Specifies the fields and methods of the objects
  - Create the objects with new
  - Classes correspond to **nouns**:  Point,Ship, HashMap

# OO REVIEW: CLASSES, METHODS, FIELDS

- A class describes a **type** of object we might create
  - Specifies the fields and methods of the objects
  - Create the objects with new
  - Classes correspond to **nouns**:  Point,Ship, HashMap
- A class has **methods**
  - **Verbs** that you can do that class: move, computeDistance

# OO REVIEW: CLASSES, METHODS, FIELDS

- A class describes a **type** of object we might create
  - Specifies the fields and methods of the objects
  - Create the objects with new
  - Classes correspond to **nouns**:  Point,Ship, HashMap
- A class has **methods**
  - **Verbs** that you can do that class: move, computeDistance
- A class has fields
  - Properties (attributes) of the class
  - A Point has an x and a y.
  - A Ship has a name (String), a location (Point), a number of crew (int)…

Has-A relationships: also called composition.

# HAS-A VS IS-A

- Has-A relationships are implemented as fields in classes
- Another kind of relationship: Is-A relationship
  - An AircraftCarrier **is-a** Ship
  - A LockFreeConcurrentHashMap **is-a** HashMap
- How do we implement is-a relationships?

# HAS-A VS IS-A

- Has-A relationships are implemented as fields in classes
- Another kind of relationship: Is-A relationship
  - An AircraftCarrier **is-a** Ship
  - A LockFreeConcurrentHashMap **is-a** HashMap
- How do we implement is-a relationships?
  - Inheritance:  public class AircraftCarrier extends Ship { … }
- How do you know whether to use is-A or has-A?
  - Brainstorm for a minute

# CHOOSING HAS-A VS IS-A

- What relationship makes sense conceptually?
  - Part of a whole -> Has-A
  - A more specific type of the same thing -> Is-A
- If you pick Is-A, would the subtype be **substitutable** for the super type?
  - Who can refresh our memory on Liskov Substitution?
- Is the interface to one type a subset of the other?
  - Yes: Is-A might be appropriate
  - No: Has-A is the only choice

# CHOOSING HAS-A VS IS-A

- For each of these choose the right relationship:
  - A Person **(has-A or is-A)** Head
  - A SportsCar **(has-A or is-A)** Car
  - A Zoo **(has-A or is-A)** List<Animal>
  - A HashTable<T> **(has-A or is-A)** Vector<T>
  - A WoodenDoor **(has-A or is-A)** PieceOfWood
  - A Shirt **(has-A or is-A)** PieceOfClothing
  - An EncryptedFileReader **(has-A or is-A)** FileReader

# CHOOSING HAS-A VS IS-A

- For each of these choose the right relationship:
  - A Person **(has-A or is-A)** Head
  - A SportsCar **(has-A or is-A)** Car
  - A Zoo **(has-A or is-A)** List<Animal>
  - A HashTable<T> **(has-A or is-A)** Vector<T>
  - A WoodenDoor **(has-A or is-A)** PieceOfWood
  - A Shirt **(has-A or is-A)** PieceOfClothing
  - An EncryptedFileReader **(has-A and is-A)** FileReader
    - This one was a bit of a trick question: it could be both! It has a FileReader to read the underlying (encrypted) data,
    - But is-A FileReader also and provides the interface of any other FileReader to give the unencrypted data

# ENCAPSULATION

- Classes provide encapsulation: combining the data with the methods that act on them
  - The data and methods are both “inside” the same object
  - We do not mess with data directly: instead, we “ask” the object to do an operation via a method
- Novice OO mistake: writing too many getters/setters
  - We often do NOT want to expose the underlying data to other classes!
    - Or even that we are using that data.

# Zoo

```
class Zoo {
   private List<Animal> animals;

}
```

- Let us think about a Zoo for a moment
  - Should we have `List<Animal> getAnimals()` and/`or void setAnimals(List<Animal>)` methods?

# ZOO

```
class Zoo {
   private List<Animal> animals;

}
```

- Let us think about a Zoo for a moment
  - Should we have `List<Animal> getAnimals()` and/`or void setAnimals(List<Animal>)` methods?
  - No and NO!
  - What to do instead?

# INSTEAD OF GET/SET… WHAT DO YOU ACTUALLY NEED?

- Instead of get/set we should think about what we actually need
  - Usually, we can think of such things in a way that HIDES the details of the internal data representation
- For example:
  - void addAnimal(Animal newAnimal)
  - void removeAnimal(Animal newAnimal)
  - Iterator<Animal> getAnimalIterator()
    - Can iterate over the Animals in this Zoo without knowing the details of how they are stored

# ZOO

```
class Zoo {
   private List<Animal> animals;
   public void addAnimal(Animal toAdd) {…}
   public void removeAnimal(Animal toRemove) { … }
   public Iterator<Animal> getAnimalIterator() { … }
}
```

- Let us suppose we had this code for Zoo…
  - Why is this better than get/set the List and letting other code use the List directly?

# ZOO

```
class Zoo {
   private List<Animal> animals;
   public void addAnimal(Animal toAdd) {…}
   public void removeAnimal(Animal toRemove) { … }
   public Iterator<Animal> getAnimalIterator() { … }
}
```

HashMap<Integer, Animal> animals;

- Let us suppose we had this code for Zoo…
  - Why is this better than get/set the List and letting other code use the List directly? **We might need to change it!**
  - Later we find that a List isn't doing what we need. So we change to a HashMap from Integer (animal ID) to Animal
    - We need to change some implementation details in Zoo
    - …but no other code should care

# STATIC (USUALLY BAD)

- Who can remind us what `static` means (same in Java and C++)?

# STATIC (USUALLY BAD)

- Who can remind us what `static` means (same in Java and C++)?
  - For a field: it is a property of the class in general, not of a particular object
  - For a method: it acts on no particular instance of the class
- Novice/intermediate OO programmer mistake: using static when you should not
  - Generally static is bad.  You should have a really good reason to use it.
  - If you need to use static, be able to articulate why

# REASONS TO USE STATIC

- “I’d really like this method to be outside any class” (use cautiously). Java requires all code to be in a class
  - Examples: Math.max, Math.sqrt
  - Should be stateless functions
    - Meaning purely a function of the inputs
- Sequentially numbering instances (field + method together)
  - E.g., BankAccount id numbers
- Private helper method to compute values to pass to super() or this() in a constructor (method only)
  - Java allows calling the parent class constructor (super()) or chaining to another constructor (this())
  - But the call must be the first line in the class. Parameters can be static calls, but not non-static calls
- Singleton pattern: we’ll discuss this later
  - Use quite sparingly---it is easily abused

# SINGLE RESPONSIBILITY + LOW COUPLING/HIGH COHESION

- We talked about the design principles of **single responsibility** and **low coupling / high cohesion.**
  - Let's take a look at some code and think about these

- First example: remember sort lines from 551?
  - Read a file
  - Sort it
  - Print the sorted results
  - (free any memory you malloced)

```c
array_t * getSortedStrings(FILE * f) {
  array_t * arrayToSort = malloc(sizeof(*arrayToSort));
  arrayToSort->array = NULL;
  arrayToSort->n = 0;
  char * curr = NULL;
  size_t sz = 0;
  size_t i = 0;
  while (getline(&curr, &sz, f) >= 0) {
    arrayToSort->array =
        realloc(arrayToSort->array, ((i + 1) * sizeof(*(arrayToSort->array))));
    arrayToSort->array[i] = curr;
    curr = NULL;
    i++;
  }
  free(curr);
  arrayToSort->n = i;
  sortData(arrayToSort->array, arrayToSort->n);
  return arrayToSort;
}

void printSorted(FILE * f) {
  array_t * sortedArray = getSortedStrings(f);
  for (size_t i = 0; i < (sortedArray->n); i++) {
    printf("%s", sortedArray->array[i]);
    free(sortedArray->array[i]);
  }
  free(sortedArray->array);
  free(sortedArray);
}
```

This code does 4 things:

1. Read input
2. Sort data
3. Print output
4. Free data

How tightly coupled are they?

```
array_t * getSortedStrings(FILE * f) {
  array_t * arrayToSort = malloc(sizeof(*arrayToSort));
  arrayToSort->array = NULL;
  arrayToSort->n = 0;
  char * curr = NULL;
  size_t sz = 0;
  size_t i = 0;
  while (getline(&curr, &sz, f) >= 0) {
    arrayToSort->array =
        realloc(arrayToSort->array, ((i + 1) * sizeof(*(arrayToSort->array))));
    arrayToSort->array[i] = curr;
    curr = NULL;
    i++;
  }
  free(curr);
  arrayToSort->n = i;
  sortData(arrayToSort->array, arrayToSort->n);
  return arrayToSort;
}

void printSorted(FILE * f) {
  array_t * sortedArray = getSortedStrings(f);
  for (size_t i = 0; i < (sortedArray->n); i++) {
    printf("%s", sortedArray->array[i]);
    free(sortedArray->array[i]);
  }
  free(sortedArray->array);
  free(sortedArray);
}
```

This code does 4 things:

1. Read input
2. Sort data
3. Print output
4. Free data

How tightly coupled are they?

1 + 2 are tightly coupled
We can do both together, but not one or the other separately

```c
array_t * getSortedStrings(FILE * f) {
  array_t * arrayToSort = malloc(sizeof(*arrayToSort));
  arrayToSort->array = NULL;
  arrayToSort->n = 0;
  char * curr = NULL;
  size_t sz = 0;
  size_t i = 0;
  while (getline(&curr, &sz, f) >= 0) {
    arrayToSort->array =
        realloc(arrayToSort->array, ((i + 1) * sizeof(*(arrayToSort->array))));
    arrayToSort->array[i] = curr;
    curr = NULL;
    i++;
  }
  free(curr);
  arrayToSort->n = i;
  sortData(arrayToSort->array, arrayToSort->n);
  return arrayToSort;
}

void printSorted(FILE * f) {
  array_t * sortedArray = getSortedStrings(f);
  for (size_t i = 0; i < (sortedArray->n); i++) {
    printf("%s", sortedArray->array[i]);
    free(sortedArray->array[i]);
  }
  free(sortedArray->array);
  free(sortedArray);
}
```

This code does 4 things:

1. Read input
2. Sort data
3. Print output
4. Free data

How tightly coupled are they?

3 +4 are tightly coupled together, and also tied to 1+2.
You can't do 3 or 4 by itself.
You can either do (1+2)
Or (1+2+3+4)

```c
array_t * readStringsFromFile(FILE * f) {
  array_t * answer = malloc(sizeof(*answer));
  answer->array = NULL;
  answer->n = 0;
  char * curr = NULL;
  size_t sz = 0;
  size_t i = 0;
  while (getline(&curr, &sz, f) >= 0) {
    answer->array = realloc(answer->array, ((i + 1) * sizeof(*(answer->array))));
    answer->array[i] = curr;
    curr = NULL;
    i++;
  }
  free(curr);
  answer->n = i;
  return answer;
}

void printArray(array_t * data) {
  for (size_t i = 0; i < (data->n); i++) {
    printf("%s", data->array[i]);
  }
}

void freeArray(array_t * data) {
  for (size_t i = 0; i < data->n; i++) {
    free(data->array[i]);
  }
  free(data->array);
  free(data);
}
```

We can split this up easily:
Separate the different behaviors.

Want to only read data?
  Call readStringsFromFile
What to sort data?
  Call sortData (not pictured)
Want to only print data?
  Call printArray
Want to free an array?
  Call freeArray

Why is this better?
Why does it matter?

```
void printSorted(FILE * f) {
  array_t * data = readStringsFromFile(f);
  sortData(data->array, data->n);
  printArray(data);
  freeArray(data);
}
```

We can still write a function to combine these behaviors as needed.  That is ok.

The important part is that we can do them separately when needed.

# DECOUPLING: COMBINING IS OK, AS LONG AS SEPARATION IS EASY

- Analogy: imagine you went to a restaurant that servers fries, burgers and shakes

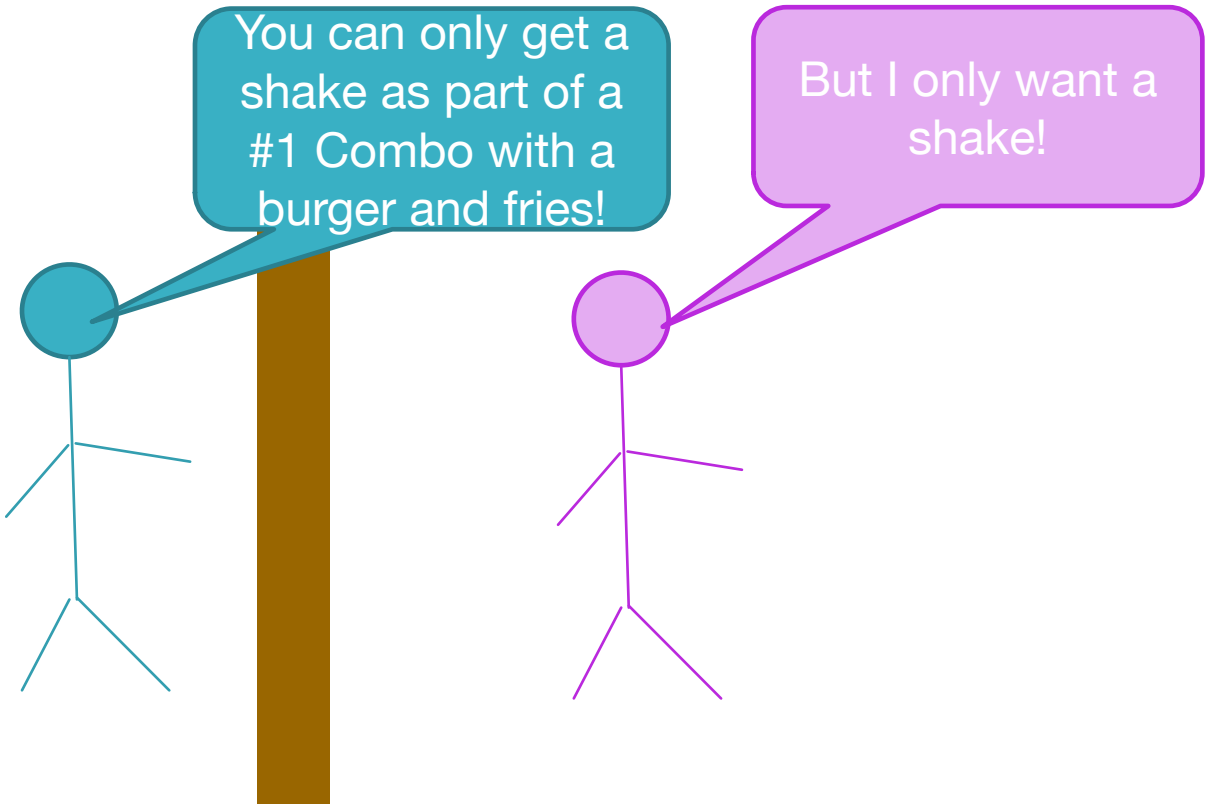
# DECOUPLING: COMBINING IS OK, AS LONG AS SEPARATION IS EASY

Combo Meal:
Fries + Burger + Shake
$8.99!


- Analogy: imagine you went to a restaurant that servers fries, burgers and shakes
  - What if they only sold them as a combo?

# DECOUPLING: COMBINING IS OK, AS LONG AS SEPARATION IS EASY

Combo Meal:
Fries + Burger + Shake
$8.99!



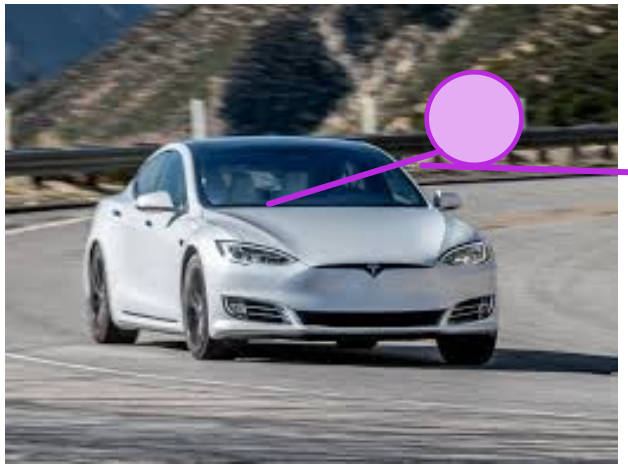

- Analogy: imagine you went to a restaurant that servers fries, burgers and shakes
  - What if they only sold them as a combo?

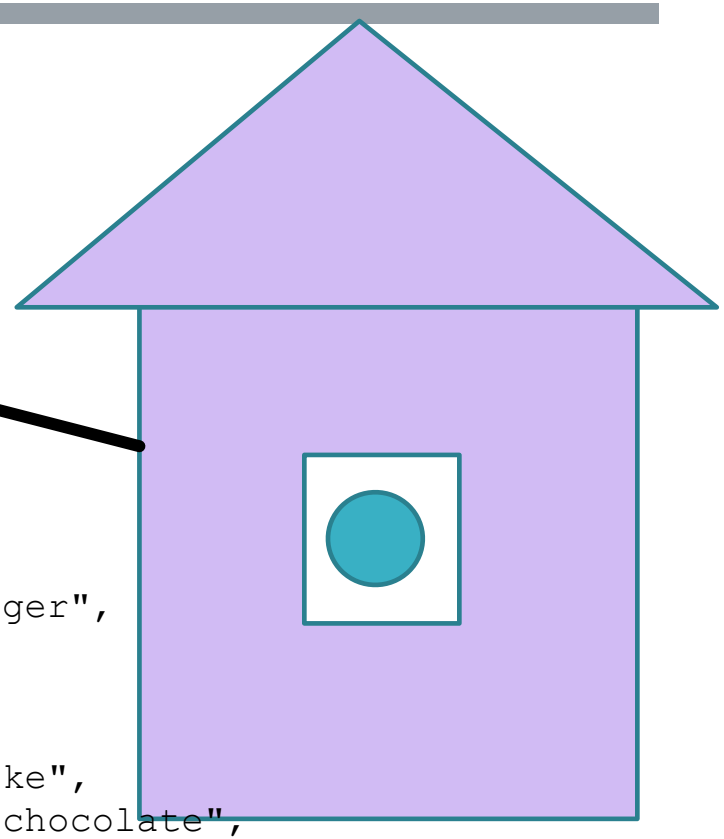
**The problem is not that there is a combo meal.**
**The problem is that you can't get the pieces separately.**

# DECOUPLING: INPUT, OPERATION, OUTPUT, CLEANUP

- Quick guidelines: always separate these tasks:
  - Input (reading files, parsing the command line, …)
  - Operation (processing the data, …)
  - Output (printing to the screen, drawing, …)
  - Cleanup (freeing memory, closing files, …)
- A little bit more to this than just not having the functions call each other
  - Let's go back to TightCoupling Burgers 'N Shakes for an example.

# BURGERS ‘N SHAKES DRIVE THRU



```
{
"items": [{
           "type": "burger",
           "count": 2
           },
           {
           "type": "shake",
           "variety": "chocolate",
           "count": 1
           }
          ],
"payment": {
           "type": "cash",
           "amount": 7.93
           }
}
```

- We add a drive through with an automated ordering system.
  - Customers push buttons, and the system sends the order in JSON format into the store (where we automate order processing)

# JSON

```
{
"items": [{
           "type": "burger",
           "count": 2
           },
           {
           "type": "shake",
           "variety": "chocolate",
           "count": 1
           }
          ],
"payment": {
            "type": "cash",
            "amount": 7.93
            }
}
```

- JSON: JavaScript Object Notation
  - Objects in curly braces
    - Field name: value
  - Arrays in []
    - Sequence of values

# JSON

```
{
"items": [{
          "type": "burger",
          "count": 2
          },
          {
          "type": "shake",
          "variety": "chocolate",
          "count": 1
          }
         ],
"payment": {
           "type": "cash",
           "amount": 7.93
           }
}
```

This is one JSON object with two fields
items
  And
payment

- JSON: JavaScript Object Notation
  - Objects in curly braces
    - Field name: value
  - Arrays in []
    - Sequence of values

# JSON

```
{
"items": [{
          "type": "burger",
          "count": 2
          },
          {
          "type": "shake",
          "variety": "chocolate",
          "count": 1
          }
         ],
"payment": {
           "type": "cash",
           "amount": 7.93
           }
}
```

The value of the items field is an array
It has two items, both of which are objects

- JSON: JavaScript Object Notation
  - Objects in curly braces
    - Field name: value
  - Arrays in []
    - Sequence of values

# JSON

```
{
"items": [{
          "type": "burger",
          "count": 2
          },
          {
          "type": "shake",
          "variety": "chocolate",
          "count": 1
          }
         ],
"payment": {
           "type": "cash",
           "amount": 7.93
           }
}
```

The first object has a “type” field, whose value is the string “burger” and a “count” field whose value is 2

(i.e., our customer ordered 2 burgers)

And so on.

- JSON: JavaScript Object Notation
  - Objects in curly braces
    - Field name: value
  - Arrays in []
    - Sequence of values

# JSON

orderStr=

```
{
"items": [{
          "type": "burger",
          "count": 2
          },
          {
          "type": "shake",
          "variety": "chocolate",
          "count": 1
          }
         ],
"payment": {
           "type": "cash",
           "amount": 7.93
           }
}
```

```
JsonObject order = JsonReader.readObject(orderStr);
JsonArray items = order.getJsonArray("items");
for (int i = 0; i < items.size(); i++) {
  JsonObject food = items.getJsonObject(i);
  //do stuff with food…
}
```

- JsonReader can turn that string into a JsonObject
  - Which has methods to get the various fields by their name (different method per type)
  - JsonArrays has size() and ability to get each item by index

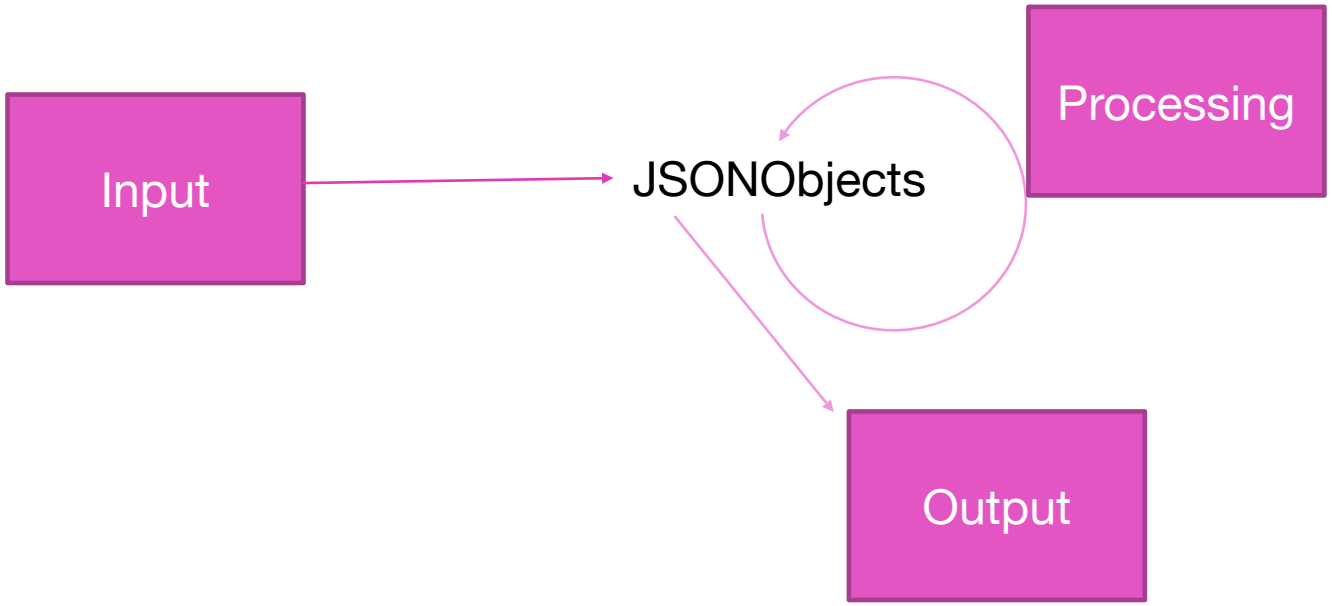
# PROCESSING TIGHTLY COUPLED TO INPUT FORMAT

```
FoodBag packOrder(String orderString){
  FoodBag fb = new FoodBag();
  JsonObject order = JsonReader.readObject(orderStr);
  JsonArray items = order.getJsonArray("items");
  for (int i = 0; i < items.size(); i++) {
     JsonObject food = items.getJsonObject(i);
     int count = food.getInt("count");
     String variety = null;
     if (food.hasKey("variety")) {
         variety = food.getString("variety");
     }
     for (int i = 0; i < count; i++) {
        fb.addFood(food.getString("type"), variety);
  }
  return fb;
}
```

**Novice temptation**:
Work directly on input format

Causes **tight coupling**
Do we like tight coupling?

# TIGHT COUPLING TO INPUT FORMAT



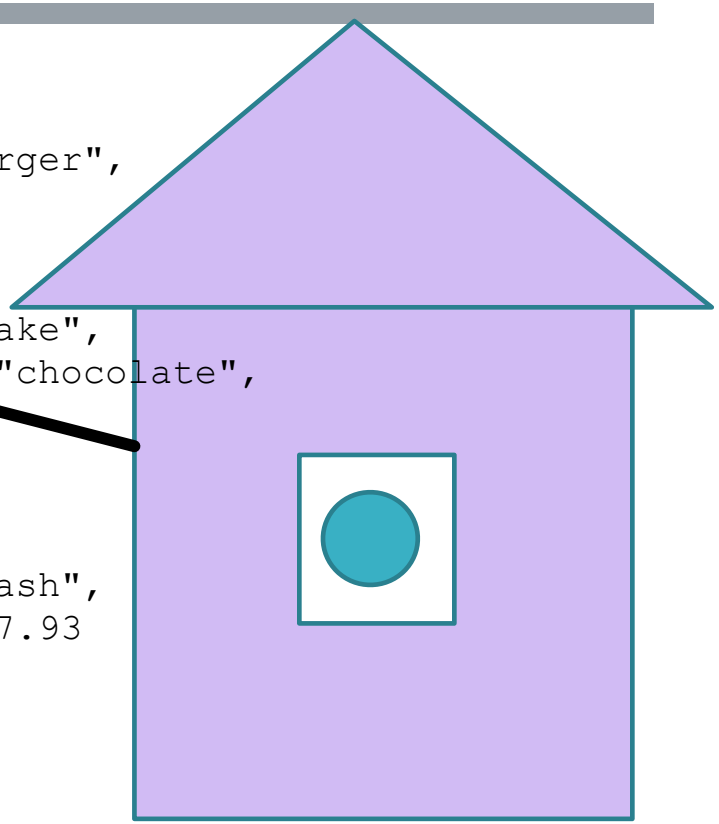

All this code operates directly on JSONObjects ☹

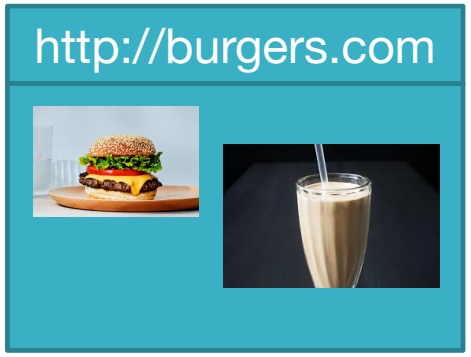
- Cumbersome
  - Objects just have getXX
  - Not even get fields, by name
  - Poor type info
- Tightly couples operations to input format
- Pushes error handling on input very late (we didn't show error handling on last slide)

BAD: Processing and output are tightly coupled with input format

# BURGERS ‘N SHAKES DRIVE THRU


```
{
"items": [{
           "type": "burger",
           "count": 2
           },
           {
           "type": "shake",
           "variety": "chocolate",
           "count": 1
           }
          ],
"payment": {
           "type": "cash",
           "amount": 7.93
           }
}
```

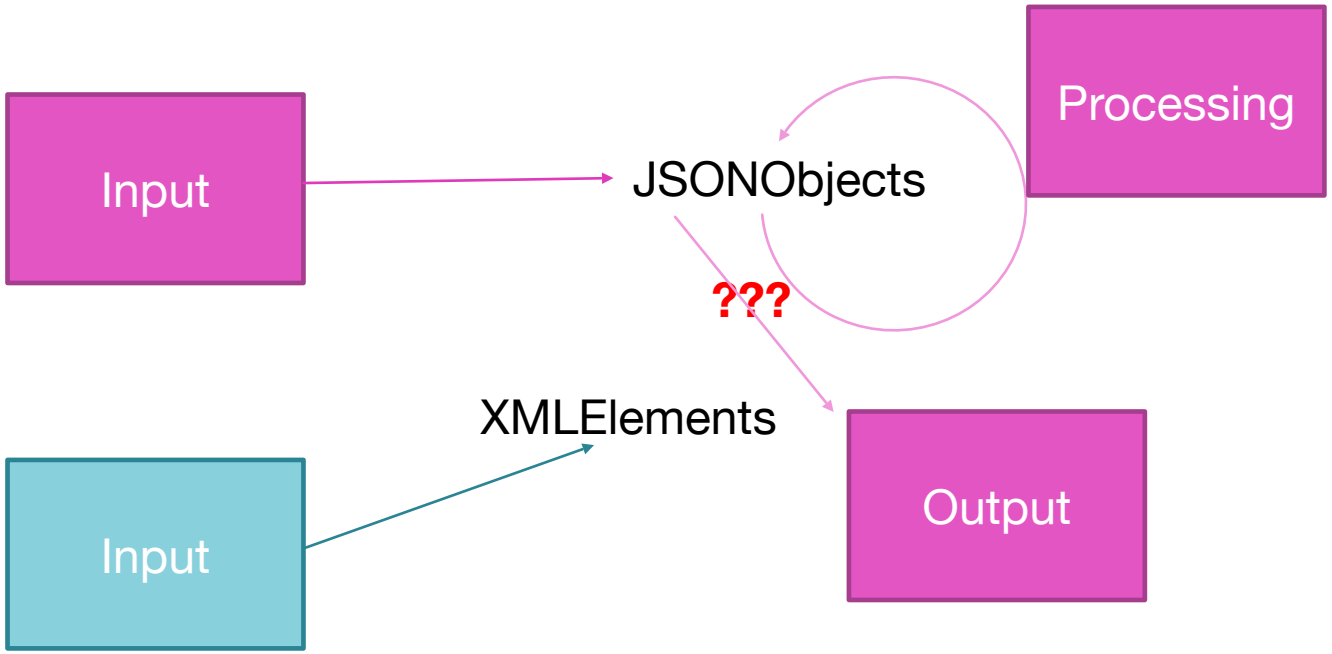





```
<order>
  <items>
    <food type="burger" count="2"/>
    <food type="shake" count="1">
       <variety name="chocolate"/>
    </food>
  </items>
  <payment type="cash" amount="7.93"/>
</order>
```

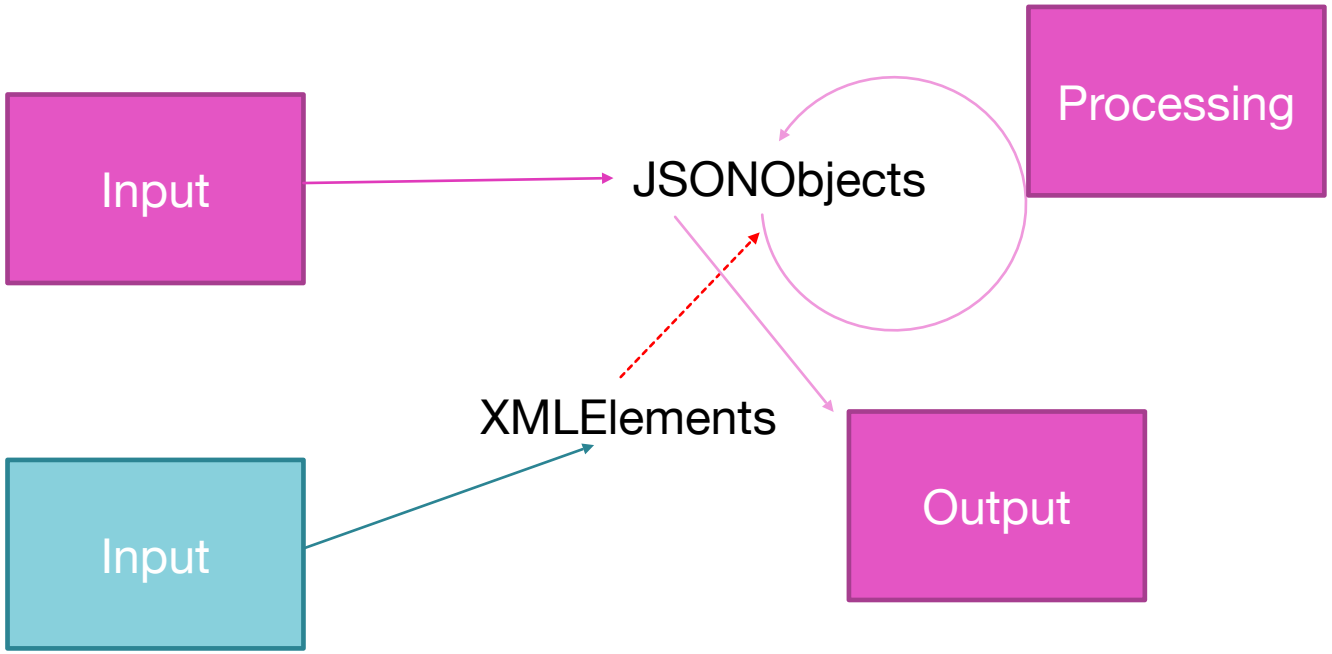
- Now we add online order, but the website sends XML orders

# DECOUPLE PROGRAM STATE FROM INPUT FORMAT



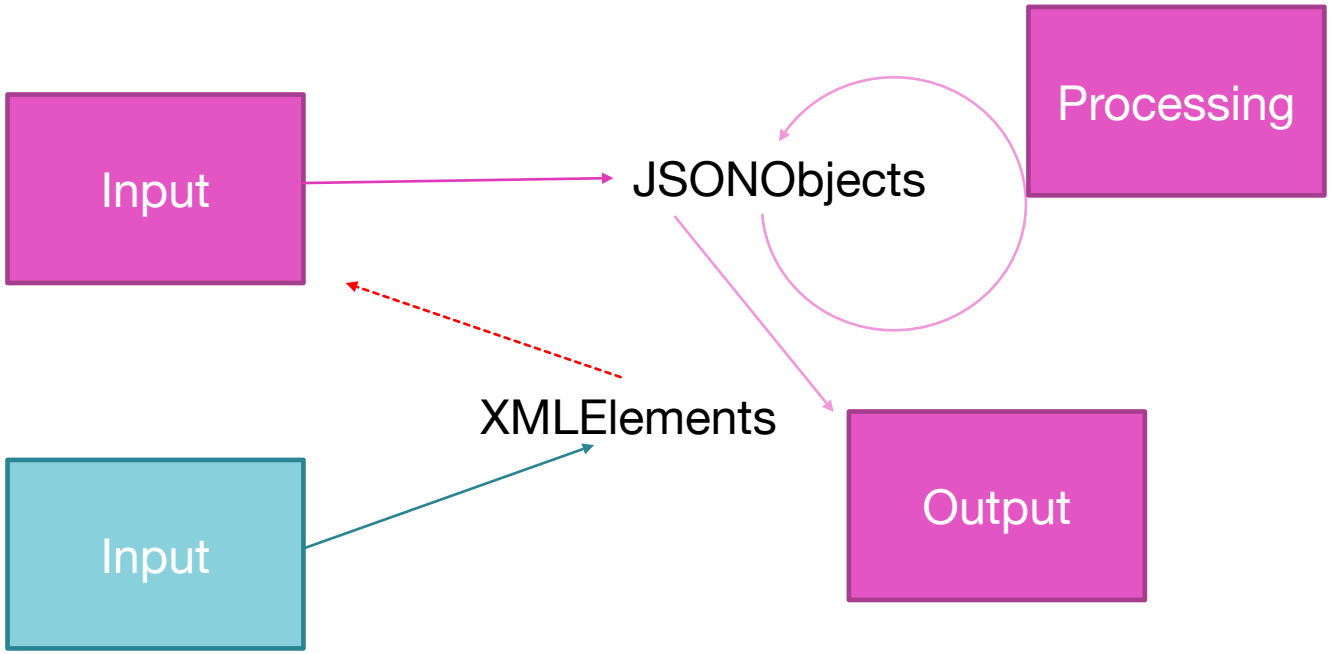

To handle this addition of web ordering, we parse the XML into XMLElements (which are not JsonObjects) and then…???

# DECOUPLE PROGRAM STATE FROM INPUT FORMAT



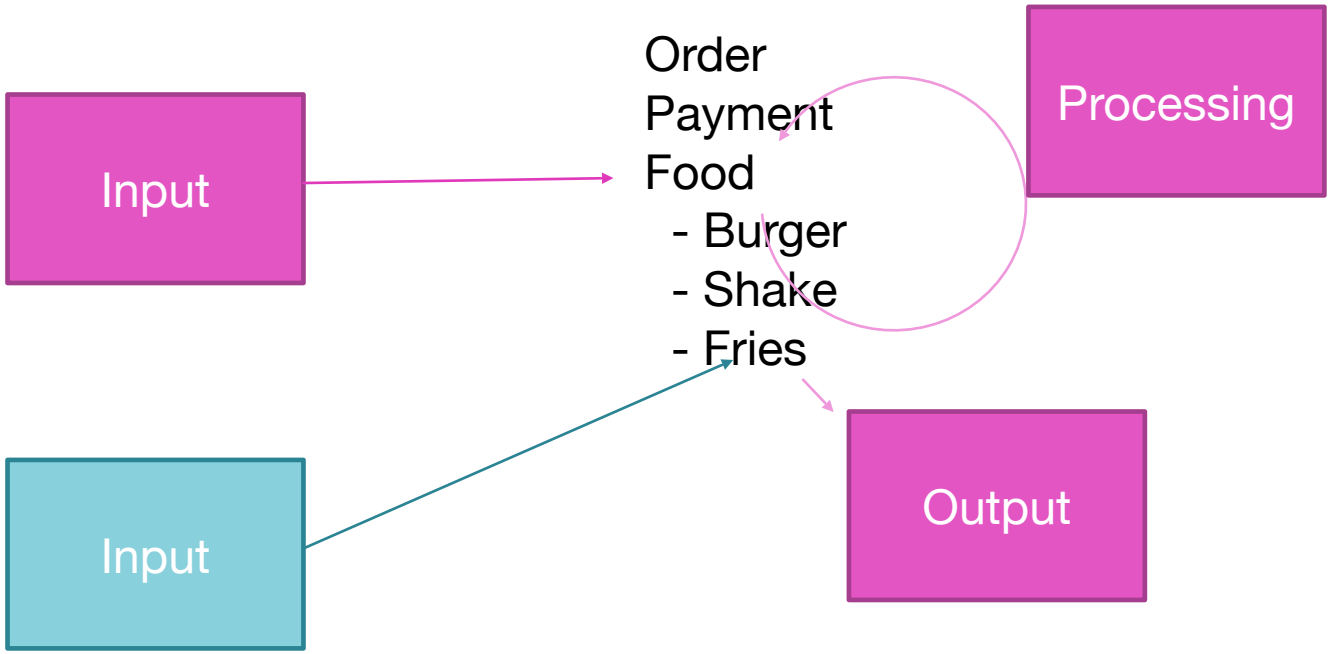

To handle this addition of web ordering, we parse the XML into XMLElements (which are not JsonObjects) and then…??? Probably write a mess of code to convert them to what we already have

# DECOUPLE PROGRAM STATE FROM INPUT FORMAT




To handle this addition of web ordering, we parse the XML into XMLElements (which are not JsonObjects) and then…???Or we might even write something to convert the input format and run it through the JSON parser..

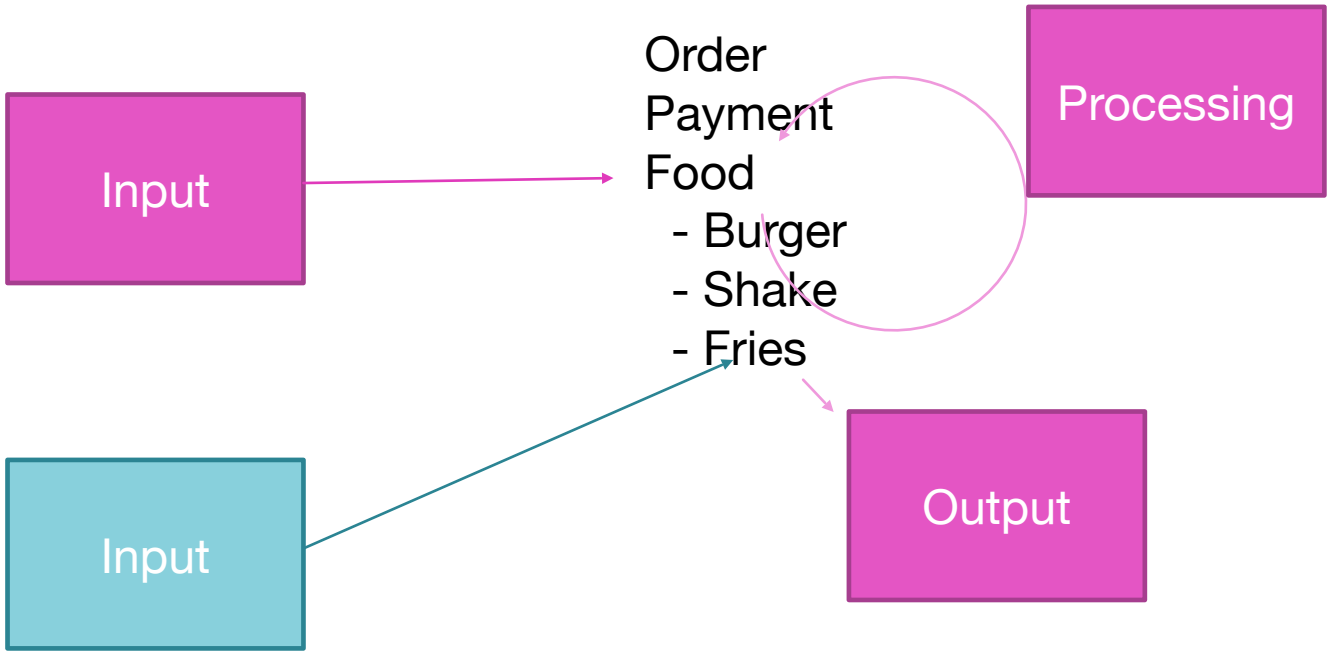
# DECOUPLE PROGRAM STATE FROM INPUT FORMAT




All of this would be much nicer if we **decoupled our program state** from our input representation!

- Work with the types we mean!
- JSON input processor produces a nice object graph of Orders, Payments, Foods, etc..
- XML input processor produces a nice object graph of Orders, Payments, Foods, etc…
- Objects can now be “smart” (more than just data holders).

# MODEL VIEW CONTROLLER

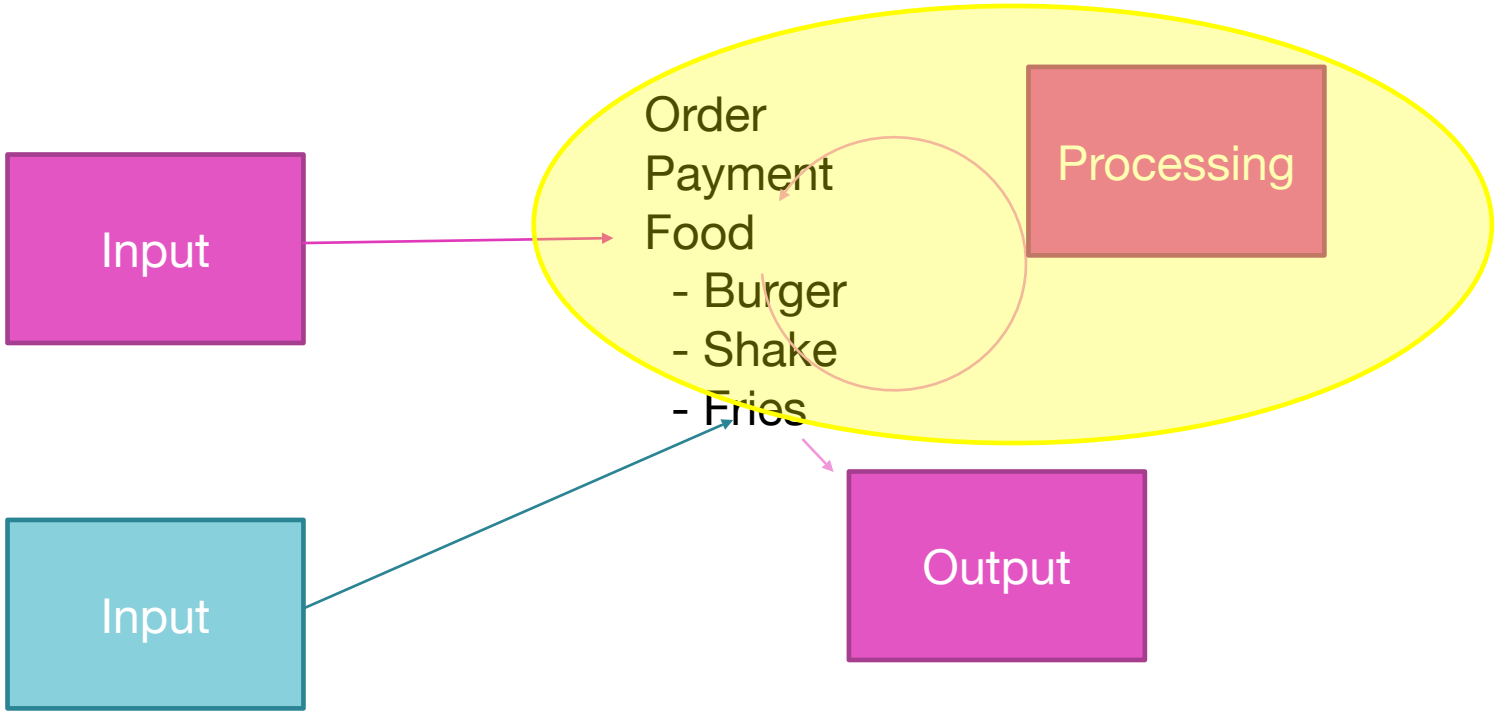



MVC: Model View Controller

Design paradigm that says we should separate the model, view and controller

What are they? What does that mean?

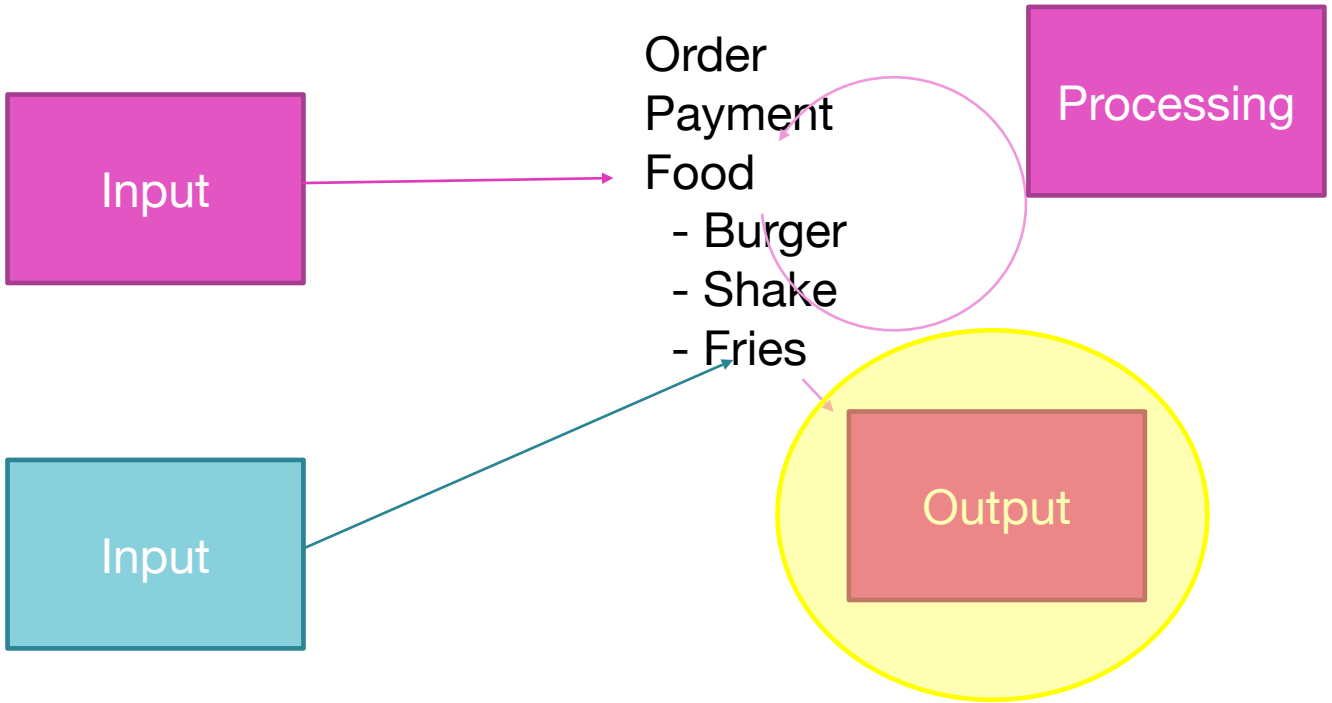
# MODEL VIEW CONTROLLER




**Model**:
The state of our program

MVC: Model View Controller
Design paradigm that says we should separate the model, view and controller
What are they? What does that mean?

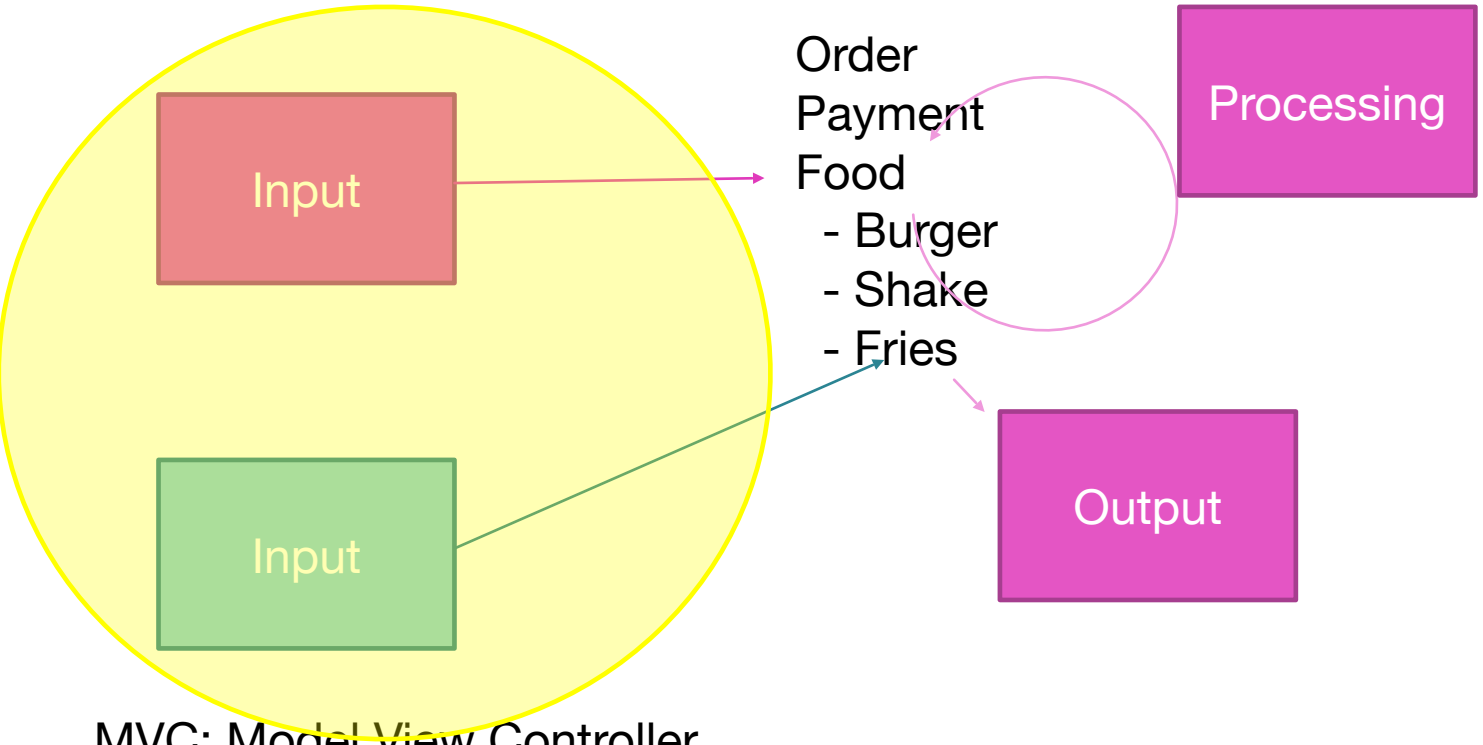
# MODEL VIEW CONTROLLER




**View**:
How we display it

MVC: Model View Controller
Design paradigm that says we should separate the model, view and controller
What are they? What does that mean?

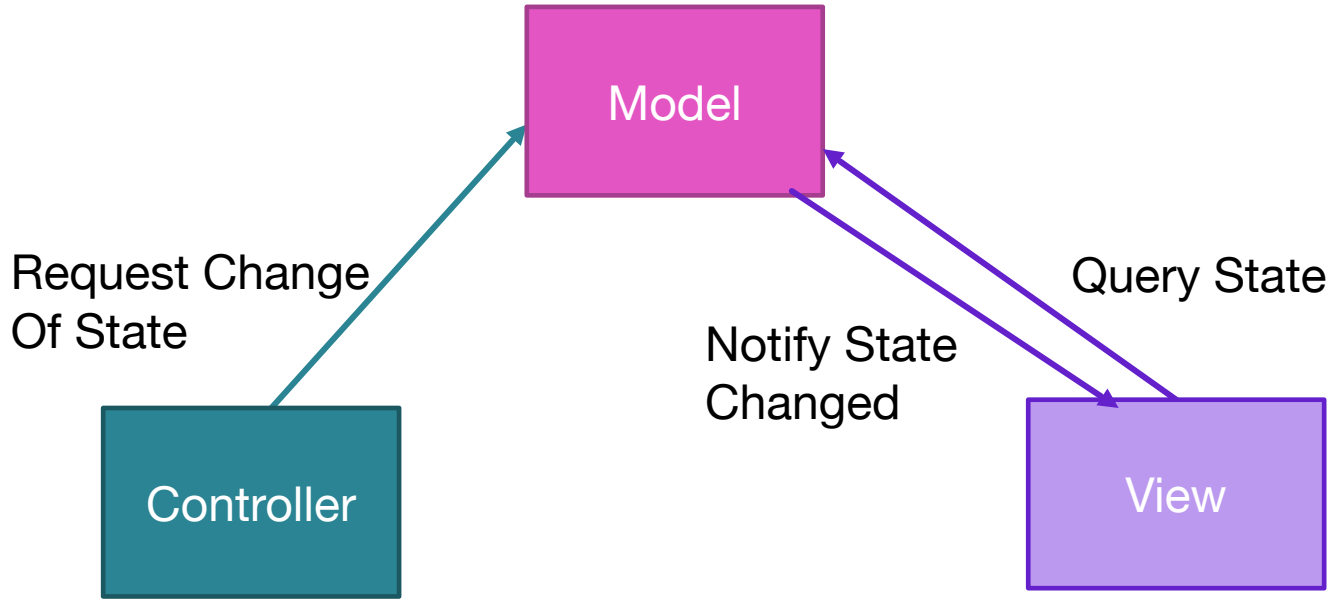
# MODEL VIEW CONTROLLER




**Controller**:
What initiates changes in the model's state

MVC: Model View Controller
Design paradigm that says we should separate the model, view and controller
What are they? What does that mean?

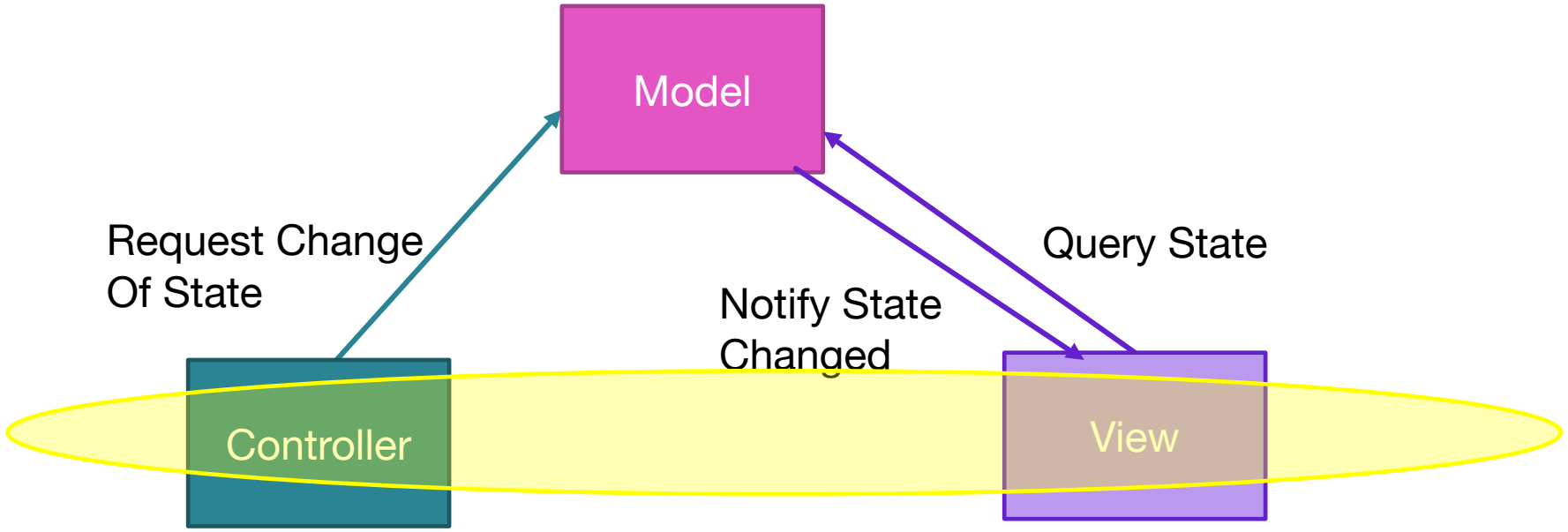
# MODEL VIEW CONTROLLER




Many novices
1: miss the point of MVC
2: apply it badly: code in wrong part, tight coupling between them
(these two things go hand in hand)

Remember that it is all about change

# MODEL VIEW CONTROLLER




**Difficulty**: View and Controller often “go together”

Text view + text input

Graphical view + graphical input

# STRINGS OFTEN ARE TIED TO INPUT (OR OUTPUT) FORMAT

- Your input is just a String, like “A3H”?
  - Battleship coordinates + orientation: Place at A3 Horizontally
- Decouple from that format ASAP!
  - Make A3 it into a Placement Object which holds a Coordinate and the orientation!
- If you need to parse it in any way (even just “index 0 is the row, index 1 is the column”…)
  - Do that in the input processing!
  - Make an object!
- We should only keep strings around when we actually need text
  - Names, descriptions, …

# ALAN PERLIS QUOTE

- “The string is a stark data structure and everywhere it is passed there is much duplication of process. It is a perfect vehicle for hiding information.”

http://www.cs.yale.edu/homes/perlis-alan/quotes.html

# DON'T TIE YOURSELF TO SPECIFIC VALUES EITHER (MOST OF THE TIME)

- Generally hardcoding values is bad
  - Computing the area of a circle?  PI isn't going to change
  - Reading input from user?  Don't' hardcode System.in ---you might want to read from somewhere else!
    - Especially for testing: we would much rather read from a ByteArrayInputStream in our unit tests
    - Pass in the InputStream to read from
    - …or OutputStream to write to.

# OBJECTS VS PRIMITIVES

- In Battleship could we make our Board by a 2D array of chars?
  - With e.g., ‘s’, ‘d’, ‘*’, ‘ ‘ etc?
- Would it be a good idea?

# PREFER OBJECTS OVER PRIMITIVES

- In Battleship could we make our Board by a 2D array of chars? Yes, but..
  - With e.g., ‘s’, ‘d’, ‘*’, ‘ ‘ etc?
- Would it be a good idea? No!
- In general, we prefer objects over primitives
  - Corollary of previous slides: get away from text + use smart objects as quickly as possible
  - If we do store our ships as letters, we must do a lot of work to reconstruct info
    - We should just keep that info in an object!
  - Example: finding the other parts of the same ship
    - May not even get it right: is this or is this

SS SS SS
SS SS SS

# PREFER OBJECTS OVER PRIMITIVES

- Instead, if we have Ship objects that know the coordinates they occupy…
  - Find which ship is at a particular coordinate?
    - Iterate through ships and ask
    - Or have a data structure that maps coordinates -> Ships

# DECOUPLING: ERROR HANDLING

- Speaking of decoupling: we should decouple error handling from the code that produces the error
  - We may wish to use the code in different situations, each with different error handling needs
- Who can remind us what OO-language construct gives us such decoupling

# EXCEPTIONS: DECOUPLE ERROR HANDLING

- Exceptions
  - Error producing code can **throw** the exception
  - Code which can handle the error uses **try**/**catch**
  - Code which cannot handle the error automatically **propagates** the exception

# 551 ERROR HANDLING: JUST EXIT

```
Placement(String descr) { //Placement constructor that takes a String
     //some code
     if (some error) {
        System.err.println("The placement was invalid");
        System.exit(1);
     }
    //other code that returns the answer
}
```

Code is tightly coupled
To error behavior:
Anything goes wrong,
program exits

- In 551, we were happy to just print an error message and exit
- Practice programming skills: detect error conditions, think about problematic cases, …
- What design problems does this approach have?

# WHAT IF THAT IS WHAT WE WANT?

- Maybe we want to print an error and exit…
  - Sometimes that is the right thing to do: can we do that in a better way?  If so, what is it/how?

# 551 ERROR HANDLING: JUST EXIT

Do I need `throws IllegalArgumentException` here?

```
Placement(String descr) {
      //some code
      if (some error) {
         throw new IllegalArgumentException("The placement was invalid");
      }
     //other code that returns the answer
}
```

- We throw an exception here
  - Some other piece of code decides how to handle it

java.lang

## Class IllegalArgumentException

java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.lang.RuntimeException
        java.lang.IllegalArgumentException

No, this is a subclass of **RuntimeException** so it is **unchecked**.

# HANDLE THE ERROR SOMEWHERE ELSE

```
Placement playTurn() {
    output.println("Where would you like to place a ship?");
    Placement p = null;
    while(p == null) {
        try {
            p = readPlacement(input); //reads line from input, does new Placement
        }
        catch(IllegalArgumentException iae) {
            output.println(iae.getMessage());
            output.println("Please try again");
        }
    }
    return p;
}
```

# TESTING

```
class PlacementTest {
  @Test
  public void test_constructor_valid() {
    Placement p1 = new Placement("A1H");   //valid placement
    assertEquals(0, p1.getCoordinate().getRow());
    assertEquals(0, p1.getCoordinate().getColumn());
    assertEquals('H', p1.getOrientation());
    //more cases here…
  }
  //should we test error cases too??
}
```

# TESTING

**assertThrows**: check that the specified Exception is thrown.
Passes if right exception type is thrown
Fails if other type (or no exception) thrown

```
class PlacementTest {
  @Test
  public void test_constructor_valid() {
    Placement p1 = new Placement("A1H");   //valid placement
    assertEquals(0, p1.getCoordinate().getRow());
    assertEquals(0, p1.getCoordinate().getColumn());
    assertEquals('H', p1.getOrientation());
    //more cases here…
  }
  @Test
  public void test_constructor_errors() {
    assertThrows(IllegalArgument.class, () -> new Placement("A0")); //missing orientation
    assertThrows(IllegalArgument.class, () -> new Placement("A0VV")); //two orientations
    assertThrows(IllegalArgument.class, () -> new Placement("0AH")); //wrong order
    //more cases here…
  }
}
```

# TESTING

The Class of the specified exception
Every class in Java has a static field .class which is the Class object corresponding to that Class.

```
class PlacementTest {
  @Test
  public void test_constructor_valid() {
    Placement p1 = new Placement("A1H");   //valid placement
    assertEquals(0, p1.getCoordinate().getRow());
    assertEquals(0, p1.getCoordinate().getColumn());
    assertEquals('H', p1.getOrientation());
    //more cases here…
  }
  @Test
  public void test_constructor_errors() {
    assertThrows(IllegalArgument.class, () -> new Placement("A0")); //missing orientation
    assertThrows(IllegalArgument.class, () -> new Placement("A0VV")); //two orientations
    assertThrows(IllegalArgument.class, () -> new Placement("0AH")); //wrong order
    //more cases here…
  }
}
```

# TESTING

An *Executable* which says what code to do (that should throw the exception).

Here (and usually): a **lambda** that executes the code we want.

```
class PlacementTest {
  @Test
  public void test_constructor_valid() {
    Placement p1 = new Placement("A1H");   //valid placement
    assertEquals(0, p1.getCoordinate().getRow());
    assertEquals(0, p1.getCoordinate().getColumn());
    assertEquals('H', p1.getOrientation());
    //more cases here…
  }
  @Test
  public void test_constructor_errors() {
    assertThrows(IllegalArgument.class, () -> new Placement("A0")); //missing orientation
    assertThrows(IllegalArgument.class, () -> new Placement("A0VV")); //two orientations
    assertThrows(IllegalArgument.class, () -> new Placement("0AH")); //wrong order
    //more cases here…
  }
}
```

Why can’t assertThrows be designed like this?

# TESTING

```
class PlacementTest {
  @Test
  public void test_constructor_valid() {
    Placement p1 = new Placement(“A1H”);   //valid placement
    assertEquals(0, p1.getCoordinate().getRow());
    assertEquals(0, p1.getCoordinate().getColumn());
    assertEquals(‘H’, p1.getOrientation());
    //more cases here…
  }
  @Test
  public void test_constructor_errors() {
    Placement p = new Placement(“A0”);
    assertThrows(IllegalArgument.class, p); //missing orientation
    assertThrows(IllegalArgument.class, () -> new Placement(“A0VV”)); //two orientations
    assertThrows(IllegalArgument.class, () -> new Placement(“0AH”)); //wrong order
    //more cases here…
  }
}
```

Why can't assertThrows be designed like this?

# TESTING

```
class PlacementTest {
  @Test
  public void test_constructor_valid() {
    Placement p1 = new Placement("A1H");   //valid placement
    assertEquals(0, p1.getCoordinate().getRow());
    assertEquals(0, p1.getCoordinate().getColumn());
    assertEquals('H', p1.getOrientation());
    //more cases here…
  }
  @Test
  public void test_constructor_errors() {
    Placement p = new Placement("A0");
    assertThrows(IllegalArgument.class, p); //missing orientation
    assertThrows(IllegalArgument.class, () -> new Placement("A0VV")); //two orientations
    assertThrows(IllegalArgument.class, () -> new Placement("0AH")); //wrong order
    //more cases here…
  }
}
```

Exception thrown here: assertThrows never executes.
We need the potentially throwing code to happen **Inside** assertThrows.

# WHAT IS EXECUTABLE (FROM ORG.JUNIT.JUPITER.API.FUNCTION)

It is an interface with one method (execute()).
The interface is tagged with **@FunctionalInterface** (says “this is for use with **lambdas**”).

```
@FunctionalInterface
public interface Executable {
  public void execute() throws Throwable;
}
```

https://docs.oracle.com/javase/8/docs/api/java/lang/FunctionalInterface.html

# WHAT GOES ON IN ASSERTTHROWS?

Runs the code
In the Executable
Inside a try-catch

```
public <T extends Throwable>
T assertThrows(Class<T> expected, Executable todo){
    try {
        todo.execute();
        fail("Did not throw any exception");
    }
    catch(Throwable t){
        assertTrue(expected.isAssignableFrom(t.getClass());
        return expected.cast(t);
    }
}
```

# WHAT GOES ON IN ASSERTTHROWS?

```
public <T extends Throwable>
T assertThrows(Class<T> expected, Executable todo){
    try {
        todo.execute();
        fail("Did not throw any exception");
    }
    catch(Throwable t){
        assertTrue(expected.isAssignableFrom(t.getClass());
        return expected.cast(t);
    }
}
```

Runs the code
In the Executable
Inside a try-catch

Checks that exception is appropriate type:
passed in type
or a subtype of it

# COULD MAKE CLASS THAT IMPLEMENTS, BUT CUMBERSOME

We could make a class that implements this interface:

```
@FunctionalInterface
public interface Executable {
   public void execute() throws Throwable;
}
class PlacementConstructorA0Executor implements Executable{
  public void execute()  {
      new Placement("A0");
  }
}
```

Then we could do

```
assertThrows(IllegalArgument.class, new PlacementConstructorA0Executor());
```

But doing so would be tediously cumbersome!

# LAMBDAS: SYNTACTIC SUGAR

Lambdas give us syntactic sugar to do this quickly and easily!

```
assertThrows(IllegalArgument.class,  ()-> new Placement("A0"));
```

Does the same things as:

Parameters to lambda = parameters to method

```
class PlacementConstructorA0Executor implements Executable{
  public void execute()  {
      new Placement("A0");
   }
}
```

.....

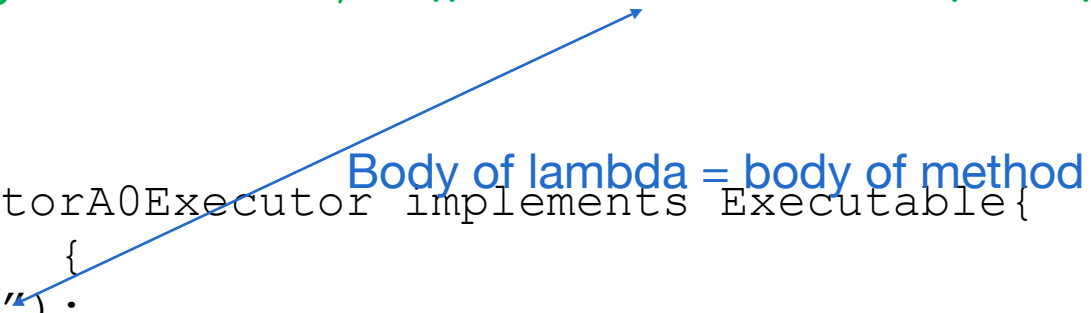
```
assertThrows(IllegalArgument.class, new PlacementConstructorA0Executor());
```

# LAMBDAS: SYNTACTIC SUGAR

Lambdas give us syntactic sugar to do this quickly and easily!

```
assertThrows(IllegalArgument.class,  ()-> new Placement("A0"));
```

Does the same things as:




```
class PlacementConstructorA0Executor implements Executable{
  public void execute()  {
      new Placement("A0");
   }
}
```

.....

```
assertThrows(IllegalArgument.class, new PlacementConstructorA0Executor());
```

See https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html for more on lambdas!

# TESTING: EXCEPTIONS

What would happen if new Placement did **`System.exit`** on error?

```
class PlacementTest {
  @Test
  public void test_constructor_valid() {
    Placement p1 = new Placement("A1H");   //valid placement
    assertEquals(0, p1.getCoordinate().getRow());
    assertEquals(0, p1.getCoordinate().getColumn());
    assertEquals('H', p1.getOrientation());
    //more cases here…
  }
  @Test
  public void test_constructor_errors() {
    assertThrows(IllegalArgument.class, () -> new Placement("A0")); //missing
orientation
    assertThrows(IllegalArgument.class, () -> new Placement("A0VV")); //two
orientations
    assertThrows(IllegalArgument.class, () -> new Placement("0AH")); //wrong order
    //more cases here…
  }
}
```

# WHAT IF WE WANT TO EXIT?

- What if we want to handle our error by exiting?
  - That may be the right thing to do…

# WHAT IF WE WANT TO EXIT?

- What if we want to handle our error by exiting?
  - That may be the right thing to do…
- We need to make that decision as “high up” as possible
  - Anything that calls the code that does System.exit cannot be used for any purpose where we don’t want to exit.
  - Main?  Ok: we don’t reuse that for other purposes
  - Depths of a library?  Likely a bad idea

# OBJECTS THAT DIFFER IN DATA VS BEHAVIOR

- When designing objects, ask whether you have a variation in **data** or **behavior**?
  - **Data**: the values of fields in the object, or parameters to methods
  - **Behavior**: the code you need to write

# OBJECTS THAT DIFFER IN DATA VS BEHAVIOR

- When designing objects, ask whether you have a variation in **data** or **behavior**?
  - **Data**: the values of fields in the object, or parameters to methods
  - **Behavior**: the code you need to write
- If your objects only differ in data, you can use the same code (thus the same class)

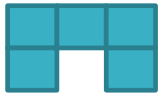
# BATTLESHIP EXAMPLE

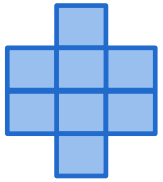
- In Battleship we have 4 types of ships
  - Submarine (represented by ‘s’) 1x2
  - Destroyer (represented by ‘d’) 1x3
  - Battleship (represented by ‘b’) 1x4
  - Carrier (represented by ‘c’) 1x6
- Do these differ in data, behavior, or both?

# BATTLESHIP EXAMPLE

- In Battleship we have 4 types of ships
  - Submarine (represented by ‘s’) 1x2
  - Destroyer (represented by ‘d’) 1x3
  - Battleship (represented by ‘b’) 1x4
  - Carrier (represented by ‘c’) 1x6
- Do these differ in data, behavior, or both?
  - **Only data-> we can write one class (RectangleShip) and use it for all four of them**
    - Name (a String)
    - Display Letter (a char)
    - Width/height (two ints)

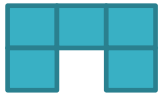
# WHAT IF WE HAVE

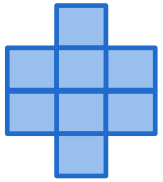




- What if we have new ships?
  - I've drawn these with colored squares, but assume they are going to be in the textual battleship
    - Each has a letter per square, etc.
  - Do these differ in **behavior** or **data** from our previous ships?

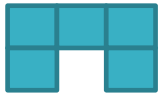
# WHAT IF WE HAVE

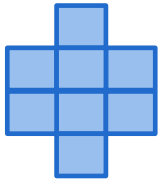




- What if we have new ships?
  - I've drawn these with colored squares, but assume they are going to be in the textual battleship
    - Each has a letter per square, etc.
  - Do these differ in **behavior** or **data** from our previous ships?
    - They are not rectangles, so we can't just make them be specific values of RectangleShip, but…
    - Could we make them the same?

# WHAT IF WE HAVE





- What if we have new ships?
  - I've drawn these with colored squares, but assume they are going to be in the textual battleship
    - Each has a letter per square, etc.
  - Do these differ in **behavior** or **data** from our previous ships?
    - They are not rectangles, so we can't just make them be specific values of RectangleShip, but…
    - Could we make them the same?
    - All of our ships are just a Set of Coordinates they occupy.
      - Just have different values in that Set.
- Goal:
  - Find ways to make differences be in data instead of in behavior.
  - Why?

Less code to write and test!

# POLYMORPHISM + DYNAMIC DISPATCH: BEHAVIOR = DATA

- Polymorphism and dynamic dispatch give us a tool for converting **behavior** differences into **data** differences
  - Call to `object.method()` inside of our code will behave differently depending on the value of `object`.
- Remember we strongly prefer dynamic dispatch over if/else
  - Open/Closed Principle
  - Decoupling