Using Java Fields

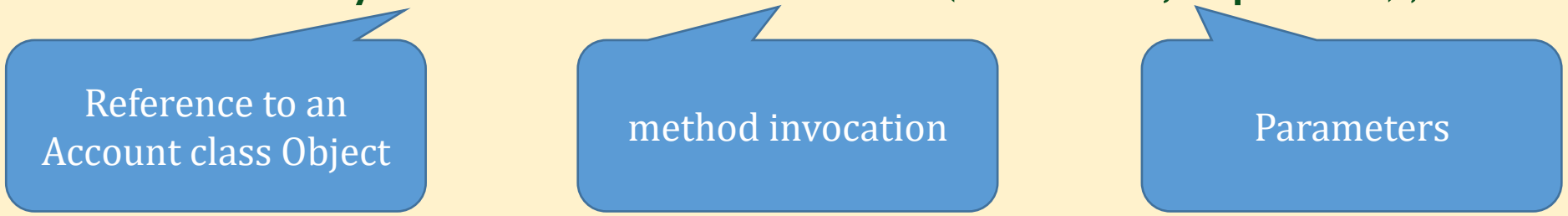
Referencing Instances of Objects

- There may be many objects of the same class instantiated at any given time
- Java keeps a "reference" to the objects to allow us to distinguish one object from another.
- References may be assigned to a "Reference Variable" – a variable used to keep a reference to a single object
- Fields are accessed by specifying: *reference_variable.field_name*

The "this" reference variable

- When you invoke a method, you always invoke "from" an instance of an object, using a reference variable e.g.

```
myAccount.writeCheck(127.32,"Sprint");
```



- *Inside* the method, the object referenced by "myAccount" is referenced by the reference variable "this"

```
if (this.balance < amount) { // overdraft
```

Implicit "this"

Inside a method, if a variable name is not a local variable, Java assumes it is a field, and automatically adds the prefix "this."

```
class Account {
    double balance;
    …
    Boolean writeCheck(double amount, String to) {
    if (this.balance >= amount) {
        this.balance -= amount;
    …
    }
}
```

```
    if (balance >= amount) {
        balance -= amount;
```

Implicit "this." added by Java

Field scope

- "Scope" – the code in which a field can be read or written to

- Field scope depends on access declaration
  - **private:** scope is the class in which the field is declared
  - **package-private:** scope is the package in which the field is declared
  - **protected:** scope is the package in which the field is declared and all sub-classes of the class in which the field is declared
  - **public:** scope is any Java code

- Note: still need a valid object reference to access an object

Class (Static) Variables

- If you use the `static` keyword, that changes a field into a ***class variable*** instead of a normal field
- Class variables have a single, global value, shared by all objects
  - Created when the class is loaded – the start of the program
  - Deleted when the class is dismissed – the end of the program
  - May be initialized when declared
  - Scope depends on access declaration, like fields
- Often used to keep track of things which a higher level class could take care of

Example Class Variable

```
class Widget {
  String wtype;  String dateMfg;
  static int count=0;
  public Widget(String t) {
    this.wtype=new String(t);
    count++;
  … }
```

Class variable declaration
Scope: package-private
Initialized to zero

```
  public static void howMany() {
    System.out.println(
    "Built " + count + " widgets."
    );
  }
}
```

Class variable reference
No implicit "this."
One variable for ALL objects

Static Variable Schematically