Inheritance
Polymorphism
Abstraction
Encapsulation

Chapter 3 Objects & Classes

information hiding - keeping things "hidden" inside of an object.

- genericly writing code helps for reuse with any type.
- inheritance allows for extending the code.
- Polymorphism allows for implementations of classes with shared logic

Class
↳ fields (data)
↳ ~~members (methods)~~
methods (functions)

Public
Private
Protected

Javadoc

@author desc
@param x desc
@throws e des
@return desc

```
/**
 * Description
 * ←
```

"The importance of proper documentation of classes can never be overstated"

Constructors
- new Class() ← Public Class() "zero param constructor"
- new Class(1,2) ← Public Class(int a, int b)
- Can be more than 1

- used to initialize the object
- defaults used if not specified

Accessors (methods)
- Does not alter state
- Single version → get state

Mutators (methods)
- changes the objects state
- Single version → set State

---

toString
- Creates string representation

@override can be overriden

equals
- checks for equality by value of objects

Chapter 3

this

- reference to current object
- works as shorthand for Constructor

- aliasing can occure when an objects reference is used on rhs & lhs

  Check:

  ```
  transfer ( Account a)
  if (a == this)
  ```

instanceof

- false if null
- true if lhs is instance of rhs

static → static members

notstatic → instance members

Static fields & methods

- Integer.parseInt()
- Math.round()
- called without instance
- can be used to sync data between objects

- Static final PI = 3.14;
- If static wasn't used each object would have copy of PI

Static Initializers

- used when initialization should happen once

```
static int[] = new int [5];
static {
  loop
}
```

- must follow field initializer

Packages

- used to organize code
- ClassPath set search dir
- classes in seperate classes have more restrictions.

Import

- Used to import package

```
import java.util.*;
```