Design by Contract
Exploiting Design Information

- Alloy provides a means for expressing properties of designs
  - Early design refinement saves time
- Ultimately, we want this effort to impact the quality of implementations
- How can we transition design information to the code?
  - State information (multiplicities, invariants, ...)
  - Operations info (pre, post, frame conditions, ...)
Design By Contract

- A method that emphasizes the precise description of interface semantics
  - not just syntax, e.g., signatures
  - but behavior, e.g., effects of a method call

- Supported by tools that
  - allow semantic properties of the design to be propagated to the code
  - support various forms of validation of those properties
Basic Idea

- Software is viewed as
  - a system of communicating components
  - all interaction is governed by contracts
  - contracts are precise specifications of mutual obligation

- Contracts are bi-directional
  - both parties are obligated by them
Specifications are Necessary

- But not sufficient for software quality
- The Law of Excluded Miracles
  - Without a specification we have little hope of a system that works properly
- Content of specification varies
  - Lightweight (partial)
  - Complete behavioral specification
- Tool support varies
  - Commercial/research, Static/dynamic checking
Contracts

- Two parties are involved in a contract
  - The supplier performs a task
  - The client requests that the task be performed
- Each party
  - has obligations
  - receives some benefits
- Contracts specify those obligations and benefits
Air Travel

Client (Traveler)
- **Obligation**
  - check in 30 minutes before boarding
  - <3 small carry-ons
  - pay for ticket
- **Benefit**
  - reach destination

Supplier (Airline)
- **Obligation**
  - take traveler to destination
- **Benefit**
  - don’t need to wait for late travelers
  - don’t need to store arbitrary amounts of luggage
  - money
Contracts

- Specify what should be done
  - they are implementation independent

- This same idea can be applied to software using the building blocks we’ve learned
  - Pre-conditions
  - Post-conditions
  - Frame-conditions
  - Invariants
Taking a flight (Java syntax)

Class Flight {

/*@ 
  requires time < this.takeoff – 30 && 
  l.number < 3 && 
  p in this.ticketed 
  ensures \result = this.destination 
  @*/

  Destination takeFlight(Person p, Luggage l) {
  ...} 
}


Why not both?

Refinement methodology
- rather than develop signatures alone
- develop contract specification
- analyze client-supplier consistency
- fill in implementation details
- check that code satisfies contract

Natural progression from design to code
Java Example

Class Mystack {
    private Object[] e elems;
    private int top, size;

    public MyStack (int s) { ... }
    public void push (Object obj) { ... }
    public Object pop() { ... }
    public boolean isEmpty() { ... }
    public boolean isFull() { ... }
}

Java Example

/*@
  invariant top >= -1 && top < size;
@*/
Class Mystack {
  private Object[] e elems;
  private int top, size;

  ...
}

Java Example

Class Mystack {
    private Object[] elems;
    private int top, size;

    /*@
    requires s > 0;
    ensures size == s && elems != null &&
    top = -1;
    @*/
    public MyStack (int s) { ... }

    ...
}

Java Example

Class Mystack {
    private Object[] elems;
    private int top, size;
    /*@
        requires  !isFull();
        ensures  top == \old(top) + 1  &&
                 elem[top] == obj;
    @*/
    public void push (Object obj) { ... }
    ...
    public boolean isFull() { ... }
}
Java Example

Class Mystack {
    private Object[] elems;
    private int top, size;
    /*@
    requires !isEmpty();
    ensures top == \old(top) - 1 &&
    \result == elem[\old(top)];
    @*/
    public Object pop() { ... }
    ...
    public boolean isEmpty() { ... }
}
Java Example

Class Mystack {
    private Object[] elems;
    private int top, size;

    ... 

   /*@ 
    ensures \result <=> top = -1; 
    @*/ 
    public boolean isEmpty() {
        ... 
    }
}
Java Example

Class Mystack {
    private Object[] elems;
    private int top, size;
    ...
    /*@
    ensures \result <=> top = size - 1;
    @*/
    public boolean isFull() { ... }
Java Example 2

```java
import java.util.Vector;

public interface ICompany {
    public Vector getEmployees();
    public Vector getRooms();
    public void hire(IEmployee employee);
    public void move(IEmployee employee, IRoom newOffice);
    public boolean roomsAvailable();
}
import java.util.Vector;
public interface ICompany {
    public Vector getEmployees();
    public Vector getRooms();
    public boolean roomsAvailable();
    /* Contract for hire(IEmployee employee) */
    /*@ requires employee != null;
    requires !getEmployees().contains(employee); // do not employ twice
    requires !employee.hasOffice(); // does not own an office somewhere else
    requires roomsAvailable(); // there must be an office left
    ensures getEmployees().contains(employee); // added to list of employees
    ensures getRooms().contains(employee.getOffice()); // assign one of our offices
    ensures employee.hasOffice(); // office assigned
    ensures employee.getOffice().getOwner() == employee; // correct office owner?
    @*/
    public void hire(IEmployee employee);
    ...
}
Source Specifications

- Pre/post conditions
  - (Side-effect free) Boolean expressions in the host language
- What about all of the expressive power we have in, e.g., Alloy?
  - Balance power against checkability
  - Balance abstractness against language mapping
- No one right choice
  - Different tools take different approaches
Important Issues

- Contract enforcement code is executed
  - It should be side-effect free
  - If not, then contracts change behavior!
- Frame conditions
  - Explicitly mention what can change
  - Anything can change
- Failed contract conditions
  - Most approaches will abort the execution
  - How can we continue?
Contract Inheritance

- Inheritance in most OO languages
  - Sub-type can be used in place of super-type
  - Sub-type provides at least the capability of super-type

- Sub-types **weaken** the pre-condition
  - Require no more than the super-type
  - Implicit or of inherited pre-conditions

- Sub-types **strengthen** the post-condition
  - Guarantee at least as much the super-type
  - Implicit and of inherited post-conditions
  - Invariants are treated the same as post-conditions
Tool Support (Java)

- Jtest (Jcontract)
  - Commercial
- iContract
  - Free, but with lots of support tools
- JML
  - major research project
  - several freely available tools
- Jass
- ...