

Perl Persistence

What's Persistence?

- Reading/writing datafiles
- Using a database
- Saving and recovering data that existed for some previous program run

Object Persistence

- Object persistence is a little more complicated
- Our objects model the Real World
- Where things stay where you put them
- Orthogonal persistence

Problems with Persistence

- Identity
- Serialization
- Storage
- Coordination

Identity

- How to locate an object when you need it
- Can be hard!
 - many languages have no run-time way to determine a variable's name, scope, type, and memory location - and to modify it to restore a persisted object

Identity in Perl

- Package variables
 - Can be accessed dynamically by name
 - A unique and reproducible identity
- Factory pattern (hash)
 - with a unique identifier, a persisted hash retrieves the object
- Singleton pattern
 - there's only one object to load or save

Encoding

- also known as
 - Serialization
 - Marshalling
 - Freezing and Thawing

Encoding

- Most storage mechanisms only allow a sequence of bytes
- Object attributes can be complex!
- Relationships between objects (references) probably won't map to the same memory addresses next time they're loaded

Encoding in Perl

- ref can tell you the run-time type
- Use type to determine structure and convert it into a sequence of characters
- Widely available serialization modules available

Encoding in Perl

- Storable
- Data::Dumper
- Freeze::Thaw

Storable

- Very fast and extremely compact representation of perl data structures
- Provides a deep copy of the structure
- Handles blessed objects
- Can be derived from

Storable

- Past version-incompatibility problems
- Not necessarily portable across architectures
- Cannot handle CODE references

Storable

```
use Animal::Cat;  
use Storable qw(freeze thaw);  
my $cat = Animal::Cat->new( name => "Kitty");  
my $encoded = freeze($cat);  
  
my $cryocat = thaw($encoded);
```

Storable

```
use Animal::Cat;  
use Storable qw(store retrieve);  
my $cat = Animal::Cat->new( name => "Kitty");  
store ($cat, "~/petsave.st");  
  
my $cryocat = retrieve("~/petsave.st");
```

FreezeThaw

- Similar interface as Storable
- Sometimes even more compact than Storable
- Also provides object methods ->Freeze and ->Thaw in UNIVERSAL
- Less well-known and supported
- Not in standard distribution

FreezeThaw

```
use Animal::Cat;  
use FreezeThaw qw(freeze thaw);  
my $cat = Animal::Cat->new( name => "Kitty");  
my $encoded = freeze($cat);  
  
my $cryocat = thaw($encoded);
```

Data::Dumper

- Not just for dumping anymore!
- Creates perl code that can be evaluated to deserialize
- Can be tweaked to be more compact
- Slowest and least compact
- Most portable

Data::Dumper

```
use Animal::Cat;
use Data::Dumper qw(Dumper);
my $cat = Animal::Cat->new( name => "Kitty");
my $encoded = Dumper($pet);

my $cryocat = eval ($encoded);
```

Data::Dumper

- Options:

```
use Data::Dumper;  
# no whitespace  
$Data::Dumper::Indent = 0;  
# newlines  
$Data::Dumper::Indent = 1;  
# use tabs so hash values line up  
$Data::Dumper::Indent = 2;  
# use comments to indicate array indices  
$Data::Dumper::Indent = 3;  
  
$Data::Dumper::Sortkeys = 1 ; # sort hash keys
```

Data::Dumper

- Options

```
use Data::Dumper;  
  
# correct nested references  
$Data::Dumper::Purity = 1;  
# note : breaks do/require  
  
# deep copy nested references  
$Data::Dumper::DeepCopy = 1;
```

Storage

- Flat files
- Simple databases (BerkeleyDB, DB_File)
- Relational databases
- Network messages

Flat Files

```
use Animal::Cat;  
use Storable qw(store retrieve);  
my $cat = Animal::Cat->new( name => "Kitty");  
store ($pet, "~/petsave.st");  
  
my $cryocat = retrieve("~/petsave.st");
```

Flat Files

```
use Animal::Cat;
use Data::Dumper;
my $cat = Animal::Cat->new( name => "Kitty");
open STORAGE, ">petsave.dd" or die($!);
print STORAGE Dumper($cat);
close STORAGE;

my $cryocat = do("~/petsave.dd");
```

Simple Database

```
use Animal::Cat;
use Storable qw(freeze thaw);
use DB_File;

my %db;
tie %db, "DB_File", "objects.db", O_RDWR |
O_CREAT, 0640, $DB_BTREE;

my $cat = Animal::Cat->new( name => "Kitty");
$db{cat} = freeze($cat);

my $cryocat = thaw($db{cat});
```

Relational Database

- Use DBI to access database
- Use BLOB types for Storable
- Use VARCHAR or string types for Data::Dumper
- Be sure to allocate enough space
- Identity issues

Object Persistence

- Serialization modules handle data and type (bless)
- Cannot handle CODE references
- Avoid CODE attributes in persisted objects

Object Persistence

- Load the modules before thawing encoded objects
- Storable will try to load modules for objects
- Whereas objects are *data* and *behavior* - only the data portion is serialized
- The *behavior* portion is encoded in your classes and loaded with the module

Object Persistence

- Data::Dumper, Storable, and FreezeThaw all provide mechanisms to hook in before freezing and thawing
- Useful to
 - delete caches
 - remove unneeded or calculable attributes
 - record data about the encoding process

Object Persistence

```
package CountFreezings;
use base qw(Storable);
sub store {
    my $self = shift @_;
    $self->{ _freezings }++;
    $self->SUPER::store(@_);
}
```

Object Persistence

```
package CountFreezings;
use Data::Dumper;
$Data::Dumper::Freezer = "prepareToFreeze";

sub prepareToFreeze {
    my $self = shift @_;
    $self->{ _freezings }++;
}

# $Data::Dumper::Toaster places a method in
the output to be evaled
```

Coarse Grained Persistence

- May lose data if program terminates by interrupt
 - # insufficient hack
`$SIG{INT} = sub { exit(0); }`
- May duplicate storage for objects referred to more than once

Fine-Grained Persistence

- Update the external representation every time the object changes
- Hard to do without modifying the classes
 - i.e. very difficult to do orthogonally
- tie can help

Fine-Grained Persistence

- Easiest way is to make the *internal* and *external* representation the same!
- Use a disk file as the object
- mmap (builtin function in perl)
- BerkeleyDB
- other databases

Other Options

- MLDBM - multi level hashes on disk
 - old and buggy - not useful for objects
 - <http://search.cpan.org/~chamas/MLDBM-2.0/lib/MLDBM.pm>
- Persistent - fairly old framework
 - easy to switch the storage layer
 - <http://www.bigsnow.org/persistent/docs/Persistent.html>

Other Options

- Class::DBI - very good ORM for Perl
 - <http://www.perl.com/pub/a/2002/11/27/classdbi.html>
- Tangram - another ORM module
 - undergoing development
 - <http://search.cpan.org/dist/Tangram/>

Homework

- `print_bills`
- `print_reminders`
- persistence with `Data::Dumper`
- Hint: regarding nested self-referential structures (like ours!)

Data::Dumper and Nesting

- Note that Data::Dumper saves data to a 'default' variable called '\$VAR1'

```
$VAR1 = [  
    bless( {  
        'pets' => [  
            bless( {  
                'name' => 'Spoon',  
                'last_grooming_date' => 0,  
                'species' => 'dog'  
            }, 'Animal::Dog' ),  
# ...  
        }, 'Owner' ),  
];
```

Data::Dumper and Nesting

- Normally self-referential structures are listed like this, which prevents eval from properly reconstructing.

```
# ...  
  
'account' => bless( {  
    'list' => [  
        bless( {  
            'charge' => '35',  
            'date' => 1111879254,  
            'service' => 'Flea dip, wash, and trim.',  
            'pet' => $VAR1[0]->{pets}[0],  
            }, 'Invoice' )  
        ],  
    }, 'InvoiceList' ),  
  
# ...
```

Data::Dumper and Nesting

- Normally self-referential structures are listed like this, which prevents eval from properly reconstructing.

```
# ...  
  
'account' => bless( {  
    'list' => [  
        bless( {  
            'charge' => '35',  
            'date' => 1111879254,  
            'service' => 'Flea dip, wash, and trim.',  
            'pet' => $VAR1[0]->{pets}[0],  
            }, 'Invoice' )  
        ],  
    }, 'InvoiceList' ),  
  
# ...
```

Data::Dumper and Nesting

- Turning on “Purity” corrects this, but now the data structure is no longer returned by do or require.

```
'account' => bless( {  
    'list' => [  
        bless( {  
            'charge' => '35',  
            'date' => 1111879254,  
            'service' => 'Flea dip, wash, and trim.',  
            'pet' => {},  
        }, 'Invoice' )  
    ],  
}, 'InvoiceList' ),  
  
# ...  
$VAR1->[0]{'account'}{'list'}[0]{'pet'} = $VAR1->[0]{'pets'}[1];  
$VAR1->[1]{'account'}{'list'}[0]{'pet'} = $VAR1->[1]{'pets'}[0];  
$VAR1->[3]{'account'}{'list'}[0]{'pet'} = $VAR1->[3]{'pets'}[0];
```

Data::Dumper

```
package PetGroomer;
use Data::Dumper;

# array of owner objects
$self->{ owners } = @owners;

open STORAGE, ">petsave.dd" or die($!);
print STORAGE Dumper( $self->{ owners } );
close STORAGE;

$self->{ owners } = do("~/petsave.dd");
```

Data::Dumper with Purity

```
package PetGroomer;
use Data::Dumper;
$Data::Dumper::Purity = 1;

# array of owner objects
$self->{ owners } = @owners;

open STORAGE, ">petsave.dd" or die($!);
print STORAGE Dumper( $self->{ owners } );
close STORAGE;
{ our $VAR1;
  do "petsave.dd";
  $self->{ owners } = $VAR1;
```

Final Project

- Use my sample data file to test `print_bills` and `print_reminders`
- Write `perldoc` for all methods in all files
- Turn in all files, including tests, to dtreder@amazon.com
- Really truly no-baloney turn-in date is about 12 noon on Saturday.
- Email me for help!