Bioconductor

and S4 Classes

Steven Buechler

Department of Mathematics
276B Hurley Hall; 1-6233

Fall, 2007

Outline

Packages Extend R

So far everything we've done is in the core *R* distribution. *R* can be extended to handle particular problems through additional packages. These aren't installed with the `R` application.

Many of the packages we'll use in microarray analysis are part of the Bioconductor system. To use a package like `affy` you must first install the package on your machine and **and** execute

```
> library(affy)
```

Packages Extend R

So far everything we've done is in the core *R* distribution. *R* can be extended to handle particular problems through additional packages. These aren't installed with the `R` application.

Many of the packages we'll use in microarray analysis are part of the Bioconductor system. To use a package like `affy` you must first install the package on your machine and **and** execute

```
> library(affy)
```

Install Bioconductor Lite Packages

Bioconductor provides a nice script for installing a default minimal set of packages. You should do this.

```
> source("http://bioconductor.org/biocLite.R")
> biocLite()
```

This installs 20 or so packages.

Install Others Through Menus

The *R* application has a menu item for installing additional packages. First browse the website to identify what you want. On your own please install the packages

`hgu133a, hgu133acdf, hgu133aprobe, hgu133plus2, hgu133plus2cdf, hgu133plus2probe`

Bioconductor calls these Metadata packages because it is data used to get annotation data about Affymetrix probes.

Additional Packages at CRAN

You may have need for *R* packages developed independently of Bioconductor, like `spBayes` for ecology. Browse the CRAN site for packages, download documentation there, and install the packages through the *R* menu.

Outline

Better Classes

to represent objects

- The classes we will encounter in Bioconductor packages are structured around an object-oriented programming paradigm S4 proposed by John Chambers (developer of *S*).
- Coding objects as lists is contrived and limits functionality. Real object-oriented standards are better, especially with large complex biological data objects.

Better Classes

to represent objects

- The classes we will encounter in Bioconductor packages are structured around an object-oriented programming paradigm S4 proposed by John Chambers (developer of *S*).
- Coding objects as lists is contrived and limits functionality. Real object-oriented standards are better, especially with large complex biological data objects.

OOP: Classes

- A class provides a software abstraction of a real world object. It reflects how we think about certain objects and what information they should contain.
- Classes are defined in terms of slots, which contain the relevant data. These are like the components in a list.
- An object is an instance of a class.
- A class defines the structure, inheritance and initialization of objects.

OOP: Classes

- A class provides a software abstraction of a real world object. It reflects how we think about certain objects and what information they should contain.
- Classes are defined in terms of slots, which contain the relevant data. These are like the components in a list.
- An object is an instance of a class.
- A class defines the structure, inheritance and initialization of objects.

OOP: Classes

- A class provides a software abstraction of a real world object. It reflects how we think about certain objects and what information they should contain.
- Classes are defined in terms of slots, which contain the relevant data. These are like the components in a list.
- An object is an instance of a class.
- A class defines the structure, inheritance and initialization of objects.

OOP: Classes

- A class provides a software abstraction of a real world object. It reflects how we think about certain objects and what information they should contain.
- Classes are defined in terms of slots, which contain the relevant data. These are like the components in a list.
- An object is an instance of a class.
- A class defines the structure, inheritance and initialization of objects.

OOP: Classes

accessing slots

The slots in an object can be accessed in several ways. The class for microarray expression data is ExpressionSet. The slot in an ExpressionSet object containing the matrix of expression values is named exprs. If `upp1Eset` is an ExpressionSet object the exprs slot can be accessed by any one of the following.

- `upp1Eset@exprs`
- `exprs(upp1Eset)`, or
- `slot(upp1Eset, "exprs")`

`slotNames(upp1Eset)` lists all the slots in this object.

OOP: Methods

- A method is a function that performs an action on an object.
- Methods define how a particular function should behave depending on the class of its arguments. (The `plot()` method behaves differently when you apply it to factors or numeric vectors.)
- Methods allow computations to be adapted to particular data types; i.e., classes.
- Associated to any object is a list of the methods that can be applied to it.

OOP: Methods

- A method is a function that performs an action on an object.
- Methods define how a particular function should behave depending on the class of its arguments. (The `plot()` method behaves differently when you apply it to factors or numeric vectors.)
- Methods allow computations to be adapted to particular data types; i.e., classes.
- Associated to any object is a list of the methods that can be applied to it.

OOP: Methods

- A method is a function that performs an action on an object.
- Methods define how a particular function should behave depending on the class of its arguments. (The `plot()` method behaves differently when you apply it to factors or numeric vectors.)
- Methods allow computations to be adapted to particular data types; i.e., classes.
- Associated to any object is a list of the methods that can be applied to it.

OOP: Methods

- A method is a function that performs an action on an object.
- Methods define how a particular function should behave depending on the class of its arguments. (The `plot()` method behaves differently when you apply it to factors or numeric vectors.)
- Methods allow computations to be adapted to particular data types; i.e., classes.
- Associated to any object is a list of the methods that can be applied to it.

Getting Help

Espcially when dealing with a number of complex object classes it is important to have good documentation. Writing this is part of the standards for packages.

```
> class?"class name"
```

gives information about the class including slot names and methods. Note that if the class is defined in a particular package you must load it first with `library()`,

```
> library(Biobase)
> class?ExpressionSet
```

(Output suppressed)

Getting Package Help

An overview about a package can be accessed by, e.g.,

```
> package?Biobase
```

(Output suppressed) However, the written documentation will be more complete.

Older Method Help Format

It isn't always clear what methods can be applied to a class written in the old list-like format. To get a list of the methods (functions) that can be applied to an object of class "clsNm", use

```
> methods(class = "clsNm")
```

Conversely, given a generic function, "func"; i.e., one written in the base *R* application, a list of the classes to which it can be applied can be found with

```
> methods("func")
```

You can see, e.g., all the classes that can be used in a `plot`.

Outline

Learn About the ExpressionSet Class

Read ExpressionSetIntroduction.pdf, documentation in the Biobase package available online.