

## Topic 8: Graphical user interfaces and file-maintenance applications

1. Class and application design
2. Inheritance and polymorphism
3. GUI construction
4. File-maintenance applications
5. Testing and correctness
6. Social and professional issues

## Features of commercial applications

- Extensive design process
- Large software development teams
- Use collections for temporary storage
- Store data longterm in disk files
- Support updating of disk data
- Use menus for user control
- Provide graphical user interfaces (GUIs)
- Extensive testing and validation

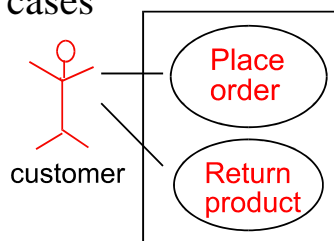
## Java applets

- Java compilers may generate *applets* that run in browser
- The browser contains a *virtual machine* that *interprets* the byte code and enables distribution of *processor independent* compiled code on the Web
- HTML tag: `<applet>`
- Programmer uses public class of applets that extends *JApplet* (package *javax.swing.JApplet*)

## 1. Class and application design

### Use cases and system specification

- *Use case*: A typical interaction between system and an *actor* in its environment
- *Actor*: A role, such as customer, manager, supplier, salesperson
- Actors initiate use cases
- Example of two use cases (UML use-case diagram):



## Elements of a robust design

- Modules (classes) may be developed, tested, deployed independently
- The design may be easily extended.
- *Example:* in an elevator simulator, migrate design easily to support multiple persons, floors, elevators

## Database management

- Supports retrievable arrangement of data
- *Examples:* library catalog, student records, many business records
- Want to separate the database management software from the data design, enabling data design on site
- Likewise *query design* is possible on site
- Simple DB management can be done with MS Word tables, Excel spreadsheets
- Effective database and query design requires *analytical thinking*

## Entity-relationship design

- The database design process may include consideration of *relationships* among *entities* stored as tables
- Example: Where *Student* and *Course* are entities, the real-world relationship “*Student registers for Course*” may guide design
- In that case, a *course-registration* table may be designed to store records whose attributes might include *Student ID*, *Course ID*, and *Date*

## Tables

- Store information for lookup
- Database tables include *metadata* (headers), may be empty
- Entities have *attributes* (fields), which have names, types, values
- Example: student table (*name, ID, email, major*)
- Row representing instance of entity is a *tuple* (record)
- *Sets* of tuples (relations) are unordered, by definition
- *Primary key* is unique identifier of a tuple

Example

## Database views

- *View*: A logical (vs. physical) table, created by an operation on a table
- Some views are a *window* into a table
- *Selection*: specification of a set of rows chosen by some criterion (e.g., all contacts with salary over \$55,000)
- *Projection*: specification of a set of columns (e.g., name and salary)

## Java packages

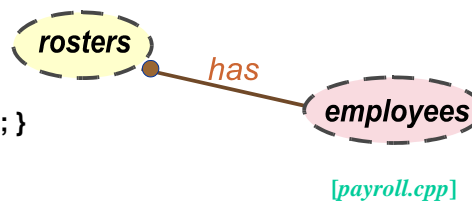
- *Package*: A set of related classes
- A package is compiled together into a single *class* file
- A package is made available to a source file using the *import* directive
- Classes and methods in packages not imported in this way may be used by listing their names before method calls

## Cohesion and coupling

- A guideline of software development practice is to maintain strong **cohesion** in a single class and weak **coupling** among different classes
- **Cohesion**: All attributes and methods are closely related to the concept implemented by the class
- **Coupling**: Dependencies among different classes. A class depends on another if it uses instances of the other class. Two valid dependencies are *containment* and *inheritance*

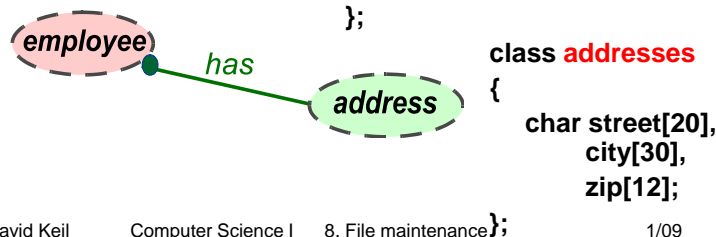
## Container class examples (C++)

```
class rosters
{
public:
    rosters() { num_recs = 0; }
private
    employees emp[100];
    int size;
};
```



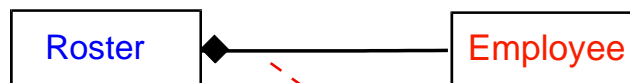
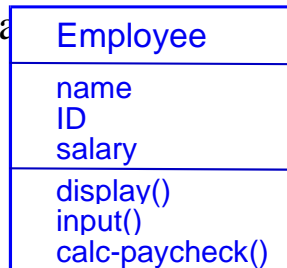
```
class employees
{
    char name[40];
    int salary;
    addresses addr;
};
```

### Collection



## UML class diagrams

- Rectangle with 3 horizontal compartments
- Class association



*Association of containment*

## 2. Inheritance and polymorphism

### Base and derived classes

- Derived class inherits members from base class
- Base class encapsulates a more general category
- Derived class = subclass = descendant
- Base class = superclass = ancestor

## Java graphics

- Packages: *javax.swing*, *java.awt*
- To create graphics window, declare *JFrame* object (example: *Emptyframeviewer.java*)
- To display an object, declare an instance and call *draw* method (example: *Rectangle*)
- To display objects, declare class to *inherit* from *JComponent*, add to frame, and call *paint* method
- To show output string *x* in message dialog:  
`JOptionPane.showMessageDialog(null, x);`
- To get input string *s* using dialog:  
`String s = JOptionPane.showInputDialog("Enter name:");`

## Extending classes with inheritance

- The concept “car” is a *kind of* vehicle in that all cars have the features of vehicles
- Cars *extend* the concept of vehicles, or *inherit* the features of vehicles; e.g., to move, steer, stop, transport cargo
- Java classes may inherit similarly, including inheriting behaviors or operations (as methods)

## Inheritance example

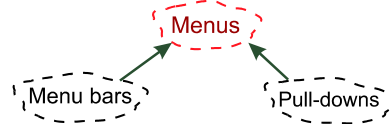
- Instances of subclasses of classes with automatic display methods inherit the auto-display behavior
- For example, a program may extend the *JComponent* class for graphical objects displayed in frames
- An instance of *JComponent* automatically displays itself when the frame is resized or moved

## Inheritance

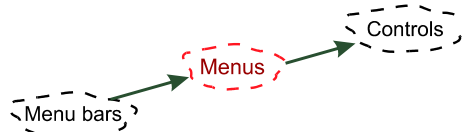
- Classes may have *kind-of* relationships by use of *extends* keyword:  
**class Employee extends Person**
- Above, *Employee* is a subclass, *Person* is a superclass, *Employee* automatically inherits all members of *Person*
- *Object* is a superclass of all other classes
- *Object* methods include
  - *clone()*
  - *toString()*
  - *equals()*

## A base class may...

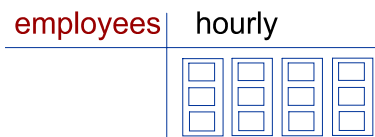
- have multiple derived classes



- be a derived class too



- have no instances



## Inner classes and anonymity

- A class may be defined inside a method definition, within another class
- This is called an *inner class*
- It is only accessible inside that “outer” class or (if the inner class is declared within a method) a method
- An anonymous object is nameless, e.g.,  $x.y$ .  
(`new(Integer(4))`)
- A class may be anonymous, e.g., declared in declaring a parameter

## Overloading operators

- Some operational concepts apply differently to different types of objects:
  - *Combine* (+) has different meanings for numbers (adding), strings (concatenating), sets (union)
  - *Compare* (>, =) has different meanings for *double*, *int*, *String*, and classes of objects
- We say that *combine* and *compare* are *overloaded*
- Overloaded operators are called *abstract*

## Interface types (classes)

- A kind of overloading implemented using the *interface* keyword
- All methods are abstract, i.e., overloaded, having names and parameters, but lacking implementation
- All methods are public but undefined (abstract)
- Interface type lacks data members (instance fields)
- To implement an interface class, declare a class that uses it and that implements its abstract methods
- *Motivation*: reusability of code

## Interface-class example

```
public interface Comparable
{
    boolean is_greater_than(Comparable x);
}
public Batter implements Comparable
{
    public boolean is_greater_than(Batter x)
    {
        return (batting_avg > x.get_batting_avg());
    }
}
```

- This shows how the Comparable interface expresses the commonality among classes like *Batter*, *Integer*, *String*, etc. – all can be compared

## Polymorphism

- When different classes implement the same interface, the different implementations reflect *polymorphism*
- Example: Different graphical objects are drawn in different ways, but all may use the same name *paint* to be drawn, where *paint* is a method of an interface class
- Polymorphism allows us to declare a collection of objects of heterogeneous types and draw all of them using the same method name *paint*

## Polymorphism uses inheritance

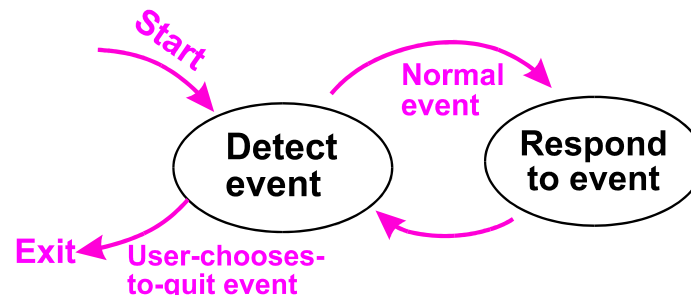
- Application programmer who uses application-class library writes a derived class that redefines the base-class event handler
- Each element of a list of base-class pointers points to a derived-class object  
*Example:* processing a mixed payroll of hourly and salaried employees

## 3. GUI construction

### Model-view-controller architecture

<b>Kind of class</b>	<b>Example</b>
<i>Model</i>	Array of database records Spreadsheet cells in linked-list grid
<i>View</i>	Window Button
<i>Controller</i>	Menu Instance of application class

## Event-driven programming



- An event is normally the user's input
- Examples of events: keypress, menu choice, mouse click

## Events in a GUI

- Event-driven programs use *event handler* code to process events such as keypresses, mouse clicks, menu choices, or passage of time
- Events are represented by objects
- *Event-listener* objects select relevant events out of all events that are generated
- *Event-source* objects generate events selected by event listeners
- HTML and JavaScript are used together to respond to some events

## Inheritance and polymorphism in GUI construction

- Specialized frame classes extend predefined *JFrame* (from *Swing* package)
- Specialized event listeners implement *ActionListener* interface
- *JTextField* components support user text input, labeled by *JLabel* objects
- *JScrollPane* objects may contain multi-line *JTextArea* objects

## Frames, buttons, and labels

- First declare button objects and labels (to identify the buttons):

```
JButton button = new JButton("Hi");  
JLabel label = new JLabel("Hi "+count++);
```
- Then create panel containing buttons
- Then define class implementing *ActiveListener*
- Listener responds to button press

## Buttons and text I/O in Java

- [See Horstmann, pp. 481-490]

## 4. File-maintenance applications

- Typical application (word processor, spreadsheet, graphics)
- Reads a file into memory
- Enables user to update file
- Saves file to disk
- Also supported:
  - Creating new document or spreadsheet
  - Saving to new file (*SaveAs*)
  - Exporting to new file format
  - Importing from different file format
  - Searching document

## File input/output

- Input uses *FileReader* and *Scanner* classes:  

```
FileReader fr = new FileReader("x.txt");  
Scanner in = new Scanner(fr);
```
- Output uses *PrintWriter*:  

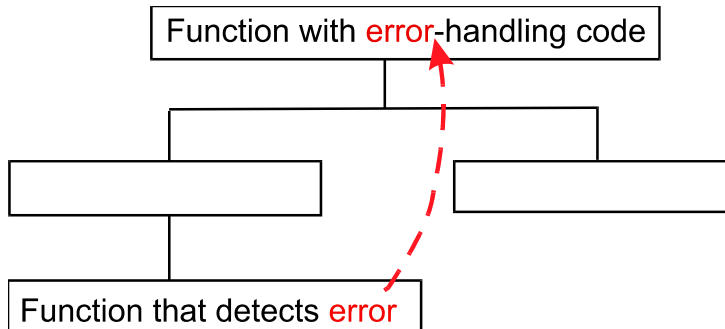
```
PrintWriter out = new PrintWriter("y.txt");
```
- If input or output file cannot be opened, a *FileNotFoundException* object will be thrown
- Methods that open files may be declared with *throws FileNotFoundException* after method header -- this will terminate method
- *JFileChooser* dialog enables user to navigate directory to choose file

## File exceptions

- Methods that open files should have *throws FileNotFoundException* at end of header; called should handle exception
- *Example*:

```
try {  
    FileReader fr = new FileReader("x.txt");  
    Scanner in = new Scanner(fr);  
    return Integer.parseInt(in.next());  
}  
catch(IOException exc) { ... }  
catch(NumberFormatException exc) { ... }  
catch(FileNotFoundException exc) { ... }  
catch(NoSuchElementException exc) { ... }
```
- File-closure can be assured using *finally*

## Why exception handling?



- With Java exceptions, error information can travel directly, across function boundaries, to where it is handled

## Exception handling

- *Purpose:* to communicate information about error situations to code that handles error
- Calling method, not method that detects exception, should handle them
- A *catch* response to an exception may be to throw another to its method's caller method
- *Examples:* Array boundary errors, file-not-found errors
- With throw of exception, current method terminates and control proceeds where exception is caught
- Compiler checks to ensure that every non-runtime (checked) exception thrown is caught and handled

## 5. Testing and correctness

- *Testing* can only prove the existence of faults, not their absence
- *Correcting a fault* late in development is expensive
- *Regression testing* finds errors introduced by maintenance process
- *Integration testing* determines whether separately developed modules work together
- *Engineering* applies teamwork, standards, science, and math to problem solving

## Unit and integration testing

- Either a single class may be tested (*unit testing*) or an entire application
- BlueJ and JUnit support unit testing
- With JUnit, a programmer creates a test class whenever a class is developed for an application
- *Integration testing* brings together the unit-tested components
- Test cases are reused for *regression testing* (re-testing after any code is modified)

## Formal verification

- *Preconditions*: Comments that state assumptions of a method: e.g., a precondition of a method that computes  $\text{sqrt}(x)$  is  $x \geq 0$
- *Postconditions*: Specification of code, or what the user can expect; e.g., a postcondition of  $\text{sqrt}(x)$  is that return value squared approximates  $x$
- *Loop invariants*: Assertions that hold at the beginning of a loop body throughout execution of the loop

## 6. Social and professional issues

- Ethics and malleability of IT
- Special features of network communication
- Intellectual property issues
- Privacy
- Professional ethics

## Ethics and the malleability of information technology

- J. Moor: Computer technology has the new feature of “logical malleability”
- This creates new possibilities for human activity
- “Computing is changing everything” (Bynum; p. 13)
- Information revolution is not only technological but social and ethical

## Special characteristics of network communication

- Scope (speed/immediacy; vastness of reach; interactivity)
- Anonymity (implying diminished trust)
- Reproducibility (enables harassment and violation of privacy)

(D. Johnson; p. 32)

## Intellectual property in the information age

- This (not regulation of cyberporn) is the main IT issue of the time
- Homologization of all information: bits, demographic data, gene maps
- Hence “the medium is not the message; the medium is *irrelevant*” (J. Boyle; p. 275)
- As Microsoft owns code in PCs; patents exist for some genes of humans
- Intellectual property rights have been expanded in copyright and then patent form

## Peer-to-peer file sharing

- P2P is distinguished from communications with central server
- Napster software enabled P2P exchange of files listed in a directory on a central server
- Courts shut down Napster for enabling copyright violations
- KaZaA: true P2P application
- After courts refused to shut down similar Grokster and Morpheus sites, RIAA sued 261 users, Fall 03

## Electronic publishing and intellectual property

- Electronic publishing reduces costs and risk of publication
- “Access to an overwhelming number of elements of daily life is now controlled by intellectual property law.” (S. Warwick)
- “Copyright in the United States is becoming more a tool for securing property interests than a mode of encouraging new works.” (Warwick)

## Reliability and security

- Testing can only prove incorrectness
- Correctness (fitting spec) is harder to show than safety
- Encryption
  - Sender encrypts data using key; receiver decrypts
  - Both sender and receiver must share key
  - Public-key cryptography solves this shared-key problem
- Access control
  - Mandatory (classification)
  - Role-based

## Privacy standards

- Fair Information Practices principles (OECD, 1980)
  - Limited collection
  - Quality
  - Purpose
  - Use limitation
  - Security
  - Openness
  - Participation (access)
  - Accountability

## Why IT raises privacy issues

- Ease of copying
- Ease of communication
- Ease of collecting data
- Power of processing data
- Storage capacity

## Professional ethics

*Responsibilities may exist toward*

- Customers and clients
- Coworkers, employees, employers
- Others affected by products and services

*Examples in other fields*

- Ethics of journalists
- Business ethics
- Science research

## Ethical decision making

- Who is affected? What are their rights?
- What are risks or issues?
- What are benefits?
- What actions are possible?
- What are responsibilities of actors?
- What are ethically acceptable choices?

## Ethics of IT professionals

- *Concerns*
  - Honesty
  - Privacy
  - Free expression
  - Intellectual property
  - Safety, security
- *Factor*: persons affected by IT work are often not customers of IT professional doing the work, and have no control
- Obligations include limiting risk to others

## Ethical considerations for software developers

- Costs, benefits to end users, including safety
- Effects on employer reputation, profits
- Some internal whistle blowing may be heard, helping company and whistle blower
- Some issues are worth going public or quitting over
- The issue is not always safety or quality versus profit
- To assess risk one must have sufficient expertise
- Disclosure of conflicts of interest is crucial
- Testing should be realistic and by persons independent of product development
- Maintenance of systems should be treated as professionally as initial development

### Codes of ethics for computer professionals

- Central concern: the public good, including human rights and diversity of culture
- Honesty and fairness in communication about software and related topics
- Use client or employer property only as authorized
- High quality, reasonable cost and schedule
- Respect for privacy, intellectual property
- Disclose conflicts of interest
- Address software errors
- Lifelong learning
- Honor agreements and assigned responsibilities

### Some guidelines

- Define objectives reasonably
- Involve users in design and testing
- Plan, estimate and schedule carefully
- Design for human users, validating input
- Validate components and default settings
- Speak honestly of risks and limitations

## Some ethical-choice scenarios

- A clinic for families with problems with violence wants its staffers to have laptops for home visits – issue is client privacy protection via security steps
- Designing an email system with targeted ads – issue is storage of customer data related to ads and responses to them
- Implementing a system design where demographic data is missing from input – ethical issue is related to following system specs

## More scenarios

- Testing of a safety-critical central application under deadline pressures to ship – should delivery be delayed?
- Copyright violations by installing more copies than licensed
- Requests to sell confidential information
- Conflict of interest – stakeholders should be informed
- Kickbacks – recommendations are expected to be honest opinions, not paid for
- Expert system for judicial sentencing

## References

ACM/IEEE. Software Engineering Code of Ethics.

ACM IT professionals' code of ethics.

S. Baase. *A Gift of Fire*, 3<sup>rd</sup> ed. Pearson Prentice Hall, 2008, Ch. 9.

Cay Horstmann. *Big Java*, 3<sup>rd</sup> ed. Wiley, 2008, Chs. 8-11.