



## **1.3 Resources**

## **1.4 Questions**

### **1.4.1 What is OOAD?**

**What is analysis?**

**What is design?**

### **1.4.2 What makes a design good?**

## **1.5 Eclipse tips**

During the course we will learn some features about the different tools that are either new, unique, or just plain helpful and are things you might not know.

### **1.5.1 Scrapbook pages**

Did you ever want to quickly see whether some code you were thinking about worked but didn't want to go through setting up a class, compiling and testing it, and so on. If so, then scrapbook pages are for you.

Information about scrapbook pages is in the Eclipse help under the Java Development User's Guide.

### **1.5.2 Coding standards**

Each project team is expected to adopt a set of coding standards and adhere to them. Eclipse assists you in structuring your code according to your standard. Look in the Window>Preferences, then to the Java>Code Style and Java>Editor>Templates

## **Auto checking**

See document: [pmd.sourceforge.net](http://pmd.sourceforge.net)

The PMD plug-in is installed on the lab machines' Eclipse configuration. You might want to load it onto your personal Eclipse configurations. PMD does a great job of checking not only style, but other potential problems that might bite you.

## **javadoc comments are required**

For this course, javadoc comments are required. Points will be taken off if there are comments missing from any public element in a Java file.

## **1.6 O-O and Java**

### **1.6.1 O-O Principles**

## **Encapsulation**

See also: [Classes](#)

## **Inheritance**

See also: [Subclassing & Interfaces](#)

## **Polymorphism**

See also: [Interfaces](#)

### *Dynamic binding*

## **1.6.2 How Java embodies O-O**

### **Interfaces**

See also: [Polymorphism](#)

### **Subclassing & Interfaces**

See also: [Inheritance](#)

### **Classes**

See also: [Encapsulation](#)

## **1.6.3 Classes and objects**

Classes and objects are two primary mechanisms (concepts) for any object-oriented paradigm. It is important to understand the differences between them when studying OOAD.

### **What are the differences?**

The class is like a template for objects. You use the class as a kind of cookie cutter to create as many objects as you need in your program.

Classes in some languages, like Java and C++, can have actual behavior and state that is shared by all instances. (static fields and methods).

Objects are instances of a class. Each object has a **state** that is determined by the values of its fields.

### **How are they related?**

Objects only have the properties (fields) and behavior (methods) that are defined by the class(es) from which they are created.

## **1.6.4 Java objects**

### **State**

#### *Field / variable values*

### **Behavior**

#### *Methods*

Methods are the primary behavior mechanism in Java.

### **Identity**

#### *References*

In Java, object variables are **always** references to the object, not copies of the object. You must explicitly copy any object if you want a copy of it. See the *clone* method in the Object class in the Java API documentation.

## **1.7 Chapter 1**

## **2 Class 2**

### **2.1 Today's topics**

#### **2.1.1 Packages**

Java programs of any reasonable size are organized into a set of packages. Packages have several purposes, not all specific to Java programming.

Packages are a component of the UML as well as Java.

#### **Grouping**

Packages are a grouping mechanism. It is a way to organize your code to make it easy to find.

#### **Encapsulation**

Beyond simple organization, a package can be used to encapsulate functionality. In an application, packages are used to encapsulate a set of collaborating classes that represent a distinct component. These components are also called subsystems.

#### **Interfaces**

Each package that encapsulates a component presents one or more interfaces to the outside world. All users of the component must use the interfaces.

#### **Implementation**

The subsystem is realized by an implementation that is hidden from the outside world and only accessed by interfaces. The subsystem may, in turn access other nested subsystems or external subsystems through their interfaces.

#### **Layering**

One architectural mechanism is *layering* the components of a system into well-defined layers. For example, there is a seven-layer model for networks called the [ISO/OSI model](#).

Another example is a client-[server model, or three-tier model](#). In such a system there is a package used for each part of the system. These packages contain other packages that make up components of the larger parts.

Subsystems are often layered themselves into layers containing the GUI, persistent data, and so on.

#### **Dependencies**

As we begin to understand OOAD we will begin to concern ourselves with the importance of package dependencies and look at techniques for measuring them and optimizing them.

Package A is dependent upon package B if a change in B would cause a change in A.

## Naming

For Java packages for this class, all packages will begin with the prefix: *edu.wpi.cs.cs4233*.

### 2.1.2 Object copies

#### Simple assignment

Remember that in Java, all object variables are references to the actual objects. Therefore, code like:

```
JFrame f1 = new JFrame();
Jframe f2;
...
f2 = f1;
```

means that f1 and f2 refer to the exact same object. No new object, that is a copy of f1, is created.

#### Cloning

When you want a copy of an object, you must create a copy yourself. The standard convention in Java is to use the clone() method.

There is a clone() method on the Object class and it is overridden in many other classes.

#### *Shallow cloning*

A clone is a *shallow clone* when it is a copy of another object, but only to the extent that each field of the first object is simply copied by assignment. That means that if object, A, has a field, f, that is a reference to an object, B, then the field, f, in a shallow clone of A, is a reference to the same object B as referenced by A.f.

#### *Deep cloning*

A *deep clone* is a clone that uses the clone() method on all object fields of the object being cloned. For consistency, all of the object fields should also have their clone() methods implemented as deep clones.

### 2.1.3 Collections

#### Definition

A collection is a representation of a set of elements and the behavior to manipulate those elements in a well- defined manner.

#### Characteristics

Collections differ depending upon certain characteristics. Usual characteristics for collections are:

- Size required for implementation
- Time required to locate an element
- Time required to add an element
- Time required to remove an element
- Ways that elements can be added or removed (i.e. anywhere in the collection, only in specific places like the front or back)
- Ordered or not

## **2.2 Questions**

### **2.2.1 What are packages used for?**

### **2.2.2 How are objects copied?**

### **2.2.3 What makes a good collection?**

## **2.3 Project**

### **2.3.1 Description**

This term you are going to implement the guidance mechanism for a product called RemoteMower. This product is a robot lawnmower that is able to mow a lawn up to 100' x 100' square. It is able to sense obstacles and compensate for them.

RemoteMower is powered by a rechargeable battery. A unique feature of RemoteMower is that it can detect obstacles and adjust its behavior accordingly. RemoteMower guidance unit ensures the most efficient coverage of the lawn by keeping track of what has been mown and where the obstacles are, and adjusting its mowing pattern accordingly.

The hardware engineers have already built the RemoteMower drive unit and provided an interface for the guidance software. They also hired a programmer to implement a simulator with an interface for editing test grids. All of this scaffolding will be made available for your team.

Your job is to implement the guidance software. You will be judged (graded) on the following:

- Correctness. Does your system work?
- Performance. Does the system mow a lawn efficiently. You only have so much power to use.
- Architecture. Have you described a clean, maintainable, extensible architecture? Can you identify your patterns and defend your design decisions?
- Implementation. Is your code clean, easy to understand, consistent?

### **Final competition**

We will have a "mow-off" at the end of the course that will determine the most efficient RemoteMower over a variety of conditions. The team whose software works the best will receive a really great prize, worth at least \$0.59.

***The competition has nothing to do with your final grade.***

### **2.3.2 Demo**

## **2.4 Evaluate Java Collection**

Look at the Java [Collection](#) interface for the J2SE 1.4.2 release. Evaluate the interface by organizing the methods, including constructors into two groups:

- Essential: These methods are absolutely necessary for a robust interface for any generic collection.
- Convenience: These methods could be removed without any loss of essential functionality. They are provided as a convenience to users of the collection.

For each, provide one or two sentences defending your choice.

Make sure your work has your name on it!

Bring your printed homework to class #4.

## **3 Class 3**

The course text focuses on the design and implementation of object-oriented software. However the analysis of the problem requirements is critical to achieving a good design. We will spend this class reviewing the analysis discipline and going into some detail on how to perform and evaluate analysis.

### **3.1 Questions**

#### **3.1.1 When is analysis finished?**

#### **3.1.2 What is the analysis model?**

### **3.2 Analysis**

Analysis is the process of transforming the problem statement and high-level requirements into detailed requirements that help drive the design activities.

#### **3.2.1 Inputs**

In order to perform analysis, you need to have certain information available. If you don't have them, you need to get them.

#### **Problem statement**

The problem statement is important because it describes, from the point of view of your primary customer(s), the problem they want you to solve.

There may be many problem statements for a system, each representing a problem at a different level. For the overall problem your team has to solve, it is a good idea to have one problem statement that describes exactly what problem you are to solve.

#### ***Vision***

The problem statement is often incorporated into a Vision statement that describes the problem, characteristics of a successful solution, stakeholder descriptions, and high-level requirements.

#### **Stakeholder needs**

All of the stakeholders have some vested interest in your work. Each of them has a set of *needs* that they expect you to satisfy when you deliver your software. You need to have the needs described and prioritized in order to do a good job of analysis. You want to be able to ensure that you have addressed (not necessarily agreed to satisfy) all of your stakeholders' needs.

For your project, you should talk to all of your stakeholders as necessary.

#### **3.2.2 Activities**

#### **Identify domain objects**

##### ***Project domain objects***

Class exercise: What are the domain objects for the RemoteMower system? Consider the complete system, not just the part you will be developing.

## *Textual analysis*

Analyzing the text of requirements, especially use cases, can provide a lot of insight into your problem domain and analysis model's domain objects.

Look for nouns to describe the objects, verbs to describe behavior (methods), and sometimes adjectives to describe other properties.

## **Assign responsibilities**

Once you have the objects, you begin to think about them abstractly (as classes) and begin to think about their responsibilities.

## *CRC Cards*

CRC Cards are designed to be a low-tech way of identifying the Classes, their Responsibilities, and Collaborators (other classes) that they need in order to accomplish their job. See the associated links for more information on CRC Cards.

### **Alistair Cockburn's article**

See document: [usingcrccards.html](http://usingcrccards.html)

### **CRC Tutorial**

See document: [crc\\_b](#)

## *Scenarios*

You can walk through the main scenarios of use cases and identify how the objects you have identified collaborate and what messages they send and respond to.

In UML, scenarios are best represented through sequence diagrams. You don't have to create a sequence diagram for every scenario. During your analysis, you want to walk through the basic flows of events of your key scenarios to ensure that you've found the key abstractions (objects and classes) and understand their primary responsibilities.

See the links for more information on sequence diagrams. **Note that one of the links points to an assigned reading.**

### **Randy Miller's Introduction**

See document: [j-jmod0508](#)

## **3.2.3 Outputs**

### **Analysis model**

#### *Use domain vocabulary*

When you identify the objects in your domain model, use the vocabulary that the people from that domain -- the domain experts -- use. Don't include a lot of technical jargon that has no meaning to them.

## *UML*

### **Classes**

### **Packages**

## Scenarios

### Use cases

#### *Don't forget text*

You can't do everything with diagrams. You will need to use text to describe the problem and different possible solutions.

#### *Code*

There will usually be some code developed during the analysis phase. Often, this will be some form of rapid prototype, or a GUI that you can use to walk the customer through the solution.

#### **Rapid prototype**

The rapid prototype is often done using some simple language such as Visual Basic, or a scripting language like Perl, Tcl, or Ruby.

#### Ruby

See document: [en](#)

Ruby has become very popular with software developers, especially those who like Smalltalk. It is a scripting language that has O-O features built into it. Everything in Ruby is an object.

Many feel that it is cleaner and more maintainable than Perl, while maintaining the great features like regular expressions. See the associated link for more information about Ruby.

#### **GUI story board**

If you're facile with Visual Basic, it makes a great tool for building quick GUIs that you can use to walk various stakeholders through the proposed system's functionality. You can also use GUI builders for your favorite language.

#### Eclipse GUI builders

##### *Jigloo*

See document: [jigloo](#)

Jigloo is an Eclipse plug-in from Cloudgarden. It is a fairly mature GUI builder that is able to construct GUIs using either Swing or SWT classes. Some students in CS509, spring 2004 used Jigloo very successfully.

Jigloo creates a *form* file that is used to generate the GUI code.

See the link for more information.

##### *VE*

See document: [vep](#)

The Eclipse Visual Editor (VE) is now available for Eclipse. It is similar to Jigloo, not quite as mature, and does not use form files.

VE requires the Eclipse Graphical Editing Framework (GEF) and the Eclipse Modeling Framework (EMF). More information on VE can be found on the associated link. VE should be loaded on the Eclipse in the laboratory machines.

### **3.2.4 Sections 2.1-2.4**

Read sections 2.1-2.4 of the text. You don't have to read the special topic 2.1.

## **4 Class 4**

### **4.1 Questions**

**4.1.1 Where does analysis end and design begin?**

**4.1.2 How do you create a good analysis scenario?**

**4.1.3 What are the goals of Design?**

### **4.2 Using analysis classes**

Is there any kind of cookbook for describing scenarios? As they say in the Hertz car rental commercials: "Not exactly." However there is a way of thinking about a system that will help you begin to take the objects / classes you discover and organize them in a non-optimal, but consistent way. This method is advocated by RUP, and other types of O-O processes. In fact, there is a standard set of UML stereotypes and icons that are used to represent such analysis classes.

#### **4.2.1 Analysis classes**

There are three types of analysis classes. You should be able to place every object or class that you've identified from the requirements into one of these three categories.

##### **Boundary**

Boundary classes, represented by the circle with a vertical line attached to its left side, is used to identify those points where an actor interacts with the system.

##### **Control**

The control class, represented by a circle with a clockwise arrow, represent classes that are responsible for sequencing operations, such as control of a scenario, or controllers for objects like databases, and so on.

##### **Entity**

The entity class, represented by a circle with a horizontal line tangent to its bottom, represents persistent objects. That is, the objects that the software acts upon. For example, Employee objects in a payroll system, or the Employee database.

#### **4.2.2 Voicemail example**

In class we will develop / examine parts of an analysis model using sequence diagrams drawn in Visual Paradigm for UML.

### **4.3 Design**

Design is one of the two main activities that software developers focus on. The other is implementation -- the creation of the code.

#### **4.3.1 Goals**

There are many goals to the design process. The designer must be aware of them and address all that apply as the software project progresses.

## Architecture

Architecture is a broad term that encompasses many activities. Software development organizations often have an architect, or team of architects, who are the primary high-level designers of their software systems.

We will address many architectural issues in this course.

### *What is it?*

See document: [definitions.html](#)

There are many definitions of software architecture. For our purposes, we will use the following from Booch, Rumbaugh, and Jacobson.

*An architecture is the set of significant decisions about the organization of a software system, the selection of the structural elements and their interfaces by which the system is composed, together with their behavior as specified in the collaborations among those elements, the composition of these structural and behavioral elements into progressively larger subsystems, and the architectural style that guides this organization---these elements and their interfaces, their collaborations, and their composition* (The UML Modeling Language User Guide, Addison- Wesley, 1999).

Other definitions are possible and many are found at:

<http://www.sei.cmu.edu/architecture/definitions.html#Classic>.

## Technology

During the design of a software system, or a system that has a heavy software component, you are faced with many choices about the type of technology that will be used in the problem solution. For example, if you are going to use a web-based system, how will you serve the content? What web servers are available that meet the requirements, and how will you choose the right one?

## Detailed specification

Design, in general, and specifically at the architectural level, are done at a higher level of abstraction than the code, or even individual class level. As the system architecture stabilizes, software design focuses on detailing the specifications (requirements, use cases, etc.) to a level that individual programmers can use.

When you use an iterative, incremental approach to development, your design evolves to the appropriate level of detail needed for each iteration. There is no big-bang design.

### *BDUF*

The level of detail of a detailed specification is an issue that is hotly debated today. The agile community opposes much up-front detailed design (also called big design up-front, or BDUF). Many larger systems, such as aerospace and telecommunications systems attempt to provide minutely detailed specifications before programmers work on production code.

## Reduce risk

Throughout the software development life cycle, you should be aware of the changing nature of risks that you face and focus on reducing those risks.

### *Risk list*

You should keep a risk list and review and update it often. Prioritize risks by how likely they are to occur, and how much damage they can cause if they do occur.

#### **4.4 Homework**

Do exercises 2.3 and 2.7 in the text, page 86.

This is due before the beginning of class 5.

Turning command: `/cs/bin/turnin submit cs4233 hw2 homework2.{doc,txt,pdf}`

#### **4.5 Sections 2.5-2.10**

Read sections 2.5-2.10 in the text. Pay particular attention to section 2.5.

### **5 What the heck are you talking about?**

Any time you don't understand something, it is **your responsibility** to speak up. It's perfectly reasonable at any time during a class (except when taking tests, etc.) to ask "**What the heck are you talking about?**" At that point the speaker has the responsibility to try and clarify the issues for you.