UNIT - VI
Cognitive Architecture

Introduction:

- There are three components in HCI -Human, Computer and Interaction
- Among them, human is the most important (consideration of human in design differentiates HCI from other fields)
- Design should take into account human (which resulted in user centred/participatory design approaches)
- We have learned about the model-based approaches (to reduce design time and effort)
- However, human behavior is very complex
  –We need more complex approaches, taking into account different components of human behavior
  –Such complexities are sought to be captured in integrated cognitive architectures (CA), which are found to be very useful in HCI

Human/User

- A user can be viewed as an information processing system

Information i/o

  –Input: visual, auditory, haptic
  –Output: movement/motor action
- Information stored in memory (Sensory, Short-term, Long-term)
- The information taken as input and stored are processed and applied in various ways
  –Reasoning
  –Problem solving
  –Skill
  –Error
  Also, emotion influences human capabilities

Message

- We want to model these activities of human information processor (HIP)
  Such (HIP) model can:
  –Validate understanding of ourselves
  –Inform the design of better user interfaces
  –Develop robust automated design approaches

HIP Models

- There are broadly two approaches to model HIP
  –Computational –HIP modeled using computer metaphors
  –Connectionist –Biological metaphor to model HIP; HIP as a neural network

Our focus –computational approaches

- Computational HIP models can be of two types:
  –Production systems: the information processing behavior is implemented as a set of production (IF-THEN-ELSE) rules
  –Non-production systems

Cognitive Architectures (CA)

- A broad theory of human cognition based on a wide selection of human experimental data, and implemented as a running computer simulation program
- Essentially, CAs are computational HIP models taking into account all the components of cognition
- There are many CAs developed, most of them are production systems with few non-production systems

Examples of CAs developed as production systems–Model Human Processor, Soar, EPIC, ACT-R/PM

Example of non-production CAs –Integrated Cognitive Subsystems

CA and HCI

- CAs are relevant to usability
  –They can provide an engineering perspective to usability analysis
  –Can be used to develop HCI-related applications
  –Can serve an important role in HCI as a theoretical science

CA as Usability Engineering Tool

- Engineering requires quantitative predictions
  –Helps compare alternative designs and identify design problems
- Qualitative/quantitative guidelines may not be sufficient to compare two "nearly equally good" designs
  –Need some theory/model to "compute" certain "parameters" of the designs to compare them
- Usability engineering involves similar situation
- Every design must be subjected to usability test
  –UE experts mostly rely on intuition/experience/guidelines to do so
  Ex: intuition may say interface X is faster than Y, but how much? 10%. 20%
  –Such quantitative data is required, as small savings in execution time may result in large financial savings
- Requires quantitative prediction theories
- Computational models based on CAs can provide such quantitative answers

The models can predict many useful quantities such as

–Execution time
–Error rate
–Transfer of knowledge
–Learning rates
–Memory load
–Many more performance measures

- Results may not be accurate but sufficient for comparison
- Models based on CA can be employed for evaluations in situations where traditional usability evaluation may be very costly or even impossible
  –For example, evaluation involving fighter pilots or astronauts
- CA based models can realistically mimic the human and act as "surrogate" user
- GOMS/cognitive walkthroughs etc. can also provide quantitative predictions
  –They are abstracted from some underlying architectures
  –CA based models can predict much more

CA Applications

- By definition, CAs are executables, making them suitable for several HCI-relevant applications
  –Intelligent Tutoring Systems
- The Lisp tutoring system
- By definition, CAs are executables, making them suitable for several HCI-relevant applications
  –Populating simulated worlds/situations
- Training fighter pilots
- Multi-party game
- Virtual reality

CA –Issues

- Many aspects of human behavior are yet to be accounted for
  –Error behavior
- It is difficult to mimic such behavior, which sometimes is required to evaluate usability
  –Emotion
- Recently some promising works are being done
- Simulated agents usually interact with an interface using a specification of the system
  –An active research area is to enable simulated agents access interfaces in ways that approximate the richness and limitation of human perceptual-motor capabilities
- At present, understanding and implementing CAs requires specialization
  –Usability of CA specification languages needs improvements

Model Human Processor –I (MHP)

- Contains three interacting systems:
  - Perceptual System
  - Cognitive System
  - Motor systems

  –For some tasks, systems operate in serial (pressing a key in response to a stimulus)

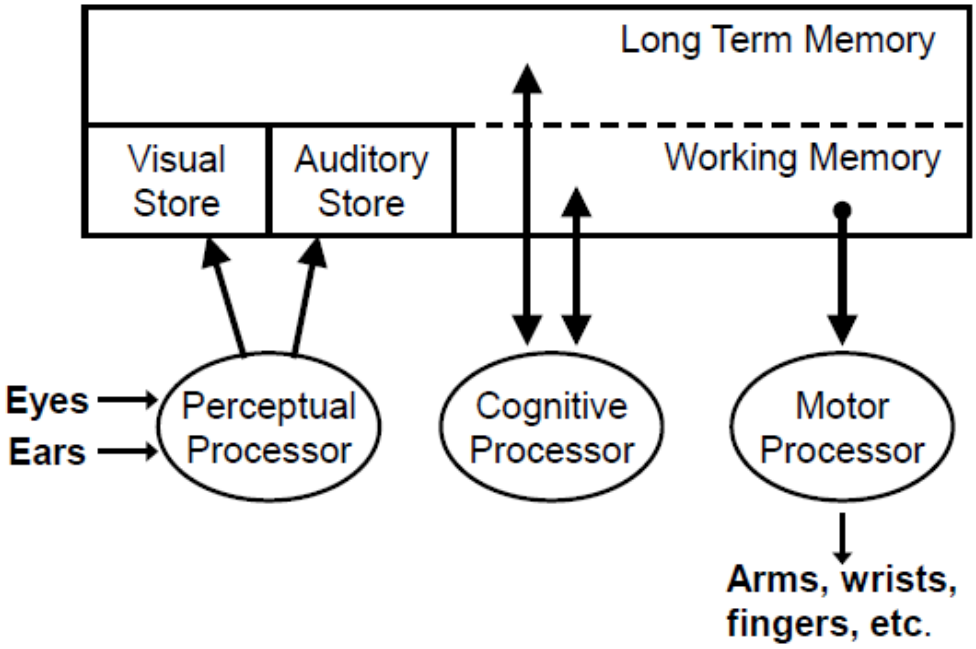
  –For other tasks, systems operate in parallel (driving, talking to passenger, listening to radio)
- Each system has its own memory and processor with characteristics

  –Memory: storage capacity and decay time

  –Processor: cycle time (includes access time)
- Each sub system in MHP guided by principles of operation

  –10 such principles in total

MHP –Schematic Diagram



Two of these principles are very important

–The rationality principle

–The problem space principle (PSP)

The Rationality Principle

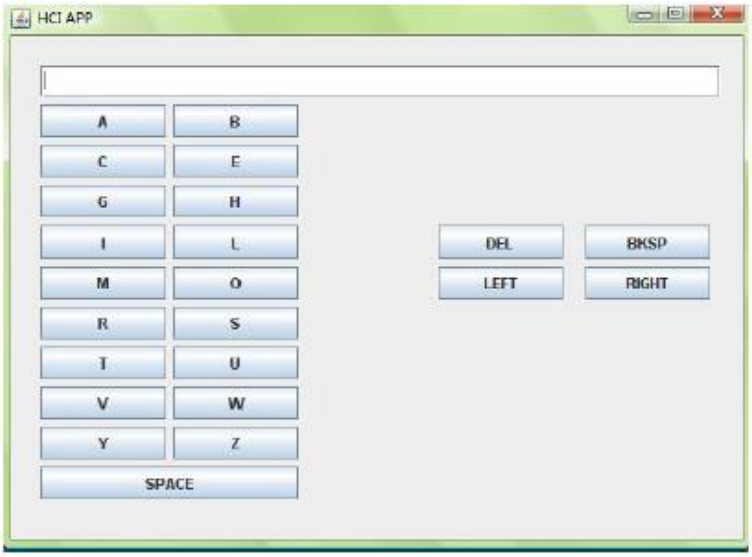
- This is one of the guiding principles of MHP
- The principle is based on the assumption that we behave rationally, rather than driven by emotion
- It states that the human behavior is determined by a set of factors that include goals, task, inputs and knowledge

The Problem Space Principle

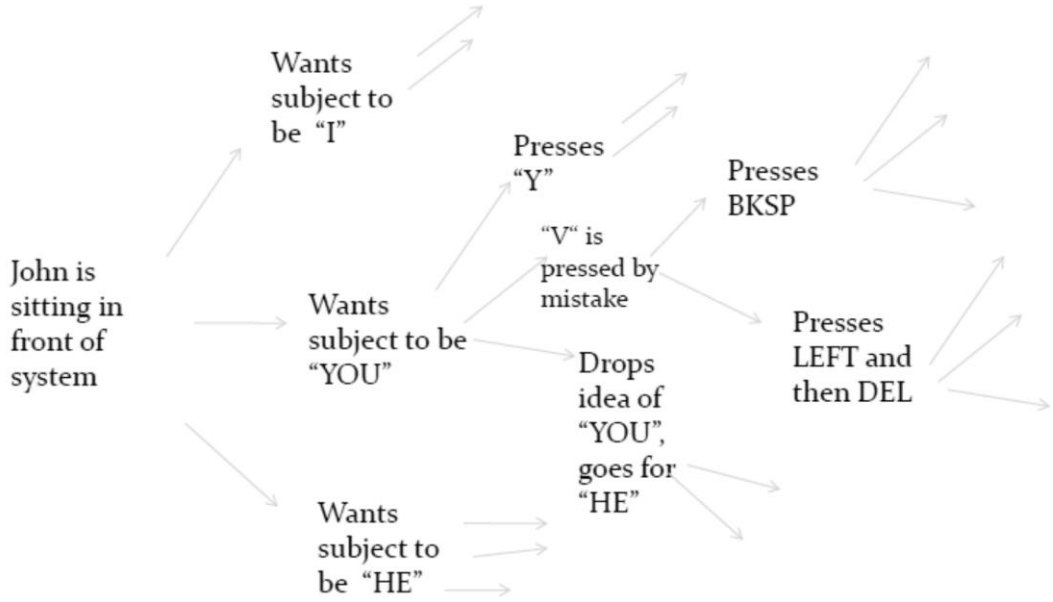
- This is the other guiding principle of MHP
- It states that any goal directed activity can be described in terms of
  –A set of states of knowledge
  –Operators for changing one state into another
  –Constraints on applying operators
  –Control knowledge for deciding which operator to apply next

A Little on PSP

- Let's understand PSP with an example. Suppose a user John wants to write a correct sentence with only available letters on the following interface



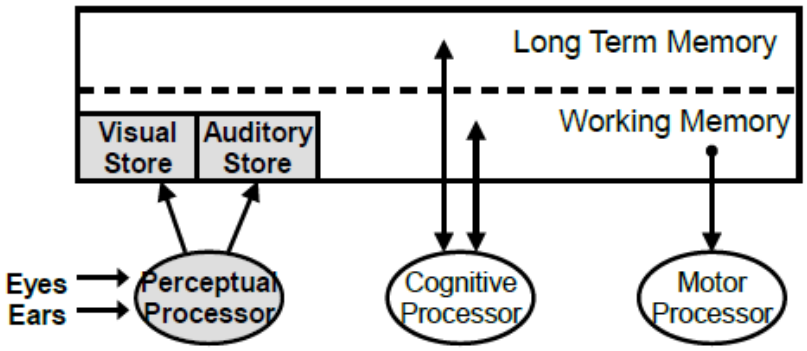
- Human cognitive behaviors assumed to have some common properties
  - Behaves in a goal-oriented manner
  - John ultimately wants to write a grammatically and semantically correct sentence
  - Operates in a rich, complex, detailed environment
  - There are many other things that John has to keep in his mind. –key positions, keystrokes
  - Uses a large amount of knowledge
  - Grammar rules, spellings
  - Behaves flexibly as a function of the environment
  - May complete any sentence left behind
  - Uses symbols and abstractions
  - John will start thinking from a greater level of abstraction
  - Learns from the environment and experience
  - As John starts typing, he learns more about available keys
- John's behavior can be described as movement through problem space
- In problem space, there are various states and by taking an appropriate action (goal directed), John reaches a new state and this process repeats until goal is achieved
- This movement in the problem space is illustrated in the diagram.



Problem space is almost a finite automata with states, and actions taken by John to change states are called operators

MHP Subsystems: Perceptual System

- Responsible for transforming external environment into a form that cognitive system can process
- Composed of perceptual memory and processor



Perceptual Memory

- Shortly after the onset of stimulus, representation of stimulus appears in perceptual memory
  –Representation is physical (non-symbolic). For example, "7" is just the pattern, not the recognized digit
- As contents of perceptual memory are symbolically coded, they are passed to the working/short term memory
- The contents of the perceptual memory gets decayed over time
- Typical decay times
  –200ms for visual store
  –1500ms for auditory store

Perceptual Processor

- The perceptual processor encodes information in the perceptual memory for about 100ms and then retrieves next stimulus
  –Cycle time = ~100ms
- Processor cannot code all information before the next stimulus arrives
  –Type and order of coding are governed by the Gestalt principles (perceive shape from atomic parts) and Attention (directs processing or filters information)

Gestalt Laws of Perception

- The organizing principles, which enables us to perceive the patterns of stimuli as meaningful wholes, are defined as
  - Proximity
  - Similarity
  - Closure
  - Continuity
  - Symmetry
  - Simplicity

Proximity

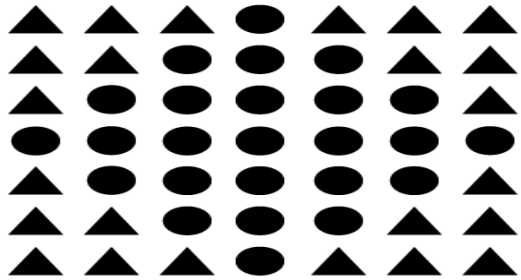
- The principle of proximity states that **things that are close together appear to be more related than things that are spaced farther apart.**

- Proximity is so powerful that it overrides similarity of color, shape, and other factors that might differentiate a group of objects.

- Notice the three groups of black and red dots above? The relative nearness of the objects has an even stronger influence on grouping than color does.
- Here, the dots appear as groups rather than a random cluster of elements

Similarity

- The principle of similarity states that **when things appear to be similar to each other, we group them together. And we also tend to think they have the same function.**
- For instance, in this image, there appear to be two separate and distinct groups based on shape: the circles and the squares.



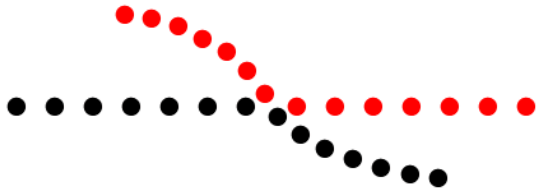
- A variety of design elements, like color and organization, can be used to establish similar groups. In the image above, for example, even though all of the shapes are the same, it's clear that each column represents a distinct group:

Closure

- The principle of closure states that **when we look at a complex arrangement of visual elements, we tend to look for a single, recognizable pattern.**
- In other words, when you see an image that has missing parts, your brain will fill in the blanks and make a complete image so you can still recognize the pattern.

Continuity

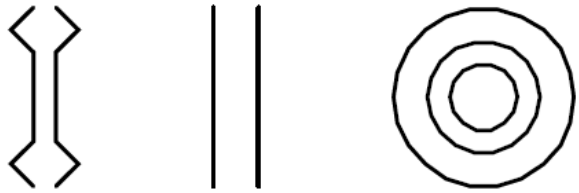
- The principle of continuity states that **elements that are arranged on a line or curve are perceived to be more related than elements not on the line or curve.**



- In the image above, for example, the red dots in the curved line seem to be more related to the black dots on the curved line than to the red dots on the straight horizontal line. That's because your eye naturally follows a line or a curve, making continuation a stronger signal of relatedness than the similarity of color.
- The stimulus appears to be made of two lines of dots, traversing each other, rather than a random set of dots

Symmetry

- The Law of Symmetry is the gestalt grouping law that states that elements that are symmetrical to each other tend to be perceived as a unified group. Similar to the law of similarity, this rule suggests that objects that are symmetrical with each other will be more likely to be grouped together than objects not symmetrical with each other. This is a lawful statement of the role of symmetry in determining figure-ground perception.



Simplicity (law of Pragnanz)

- The word *pragnanz* is a German term meaning "good figure." The law of Pragnanz is sometimes referred to as the law of good figure or the law of simplicity. This law holds that objects in the environment are seen in a way that makes them appear as simple as possible.

You see the image above as a series of overlapping circles rather than an assortment of curved, connected lines.

Principles of Perceptual System

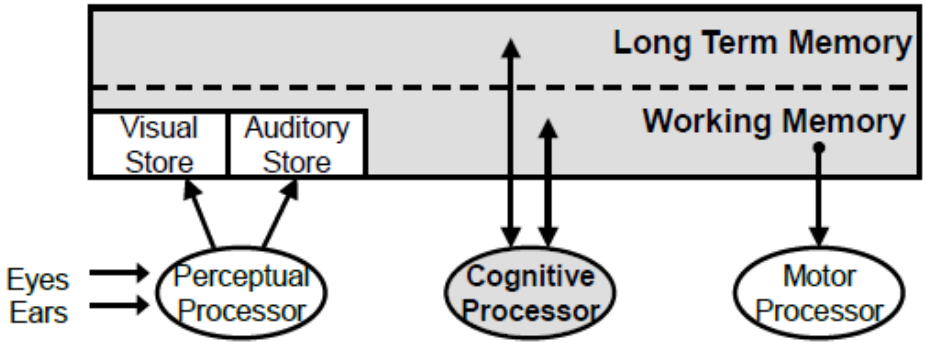
There are two principles governing the working of the perceptual system

- **Variable processor rate principle** -processor cycle time varies inversely with stimulus intensity (brighter screens need faster refresh rates)
- **Encoding specificity principle** -encoding at the time of perception impacts (a) what and how information is stored and (b) what retrieval cues are effective at retrieving the stored information

Cognitive System

- The cognitive system in MHP is responsible for decision making
- It is a production system comprising of
  –A set of production (IF-THEN) rules (stored in the memory; working memory (WM) + long term memory (LTM))
  –A rule interpretation engine (cognitive processor)
- It uses the contents of WM and LTM to make decisions and schedule actions with motor system

Composed of a processor and the two memories (WM and LTM)



Working Memory

- Holds intermediate products of thinking and representations produced by perceptual system
- Comprised of activated sections of LTM called "chunks"
  –A chunk is a hierarchical symbol structure
  –7 ±2 chunks active at any given time (known as the the 7 ±2 principle)
- The memory content gets decayed
- The decay is caused by:
  –Time: about 7s for three chunks, but high variance
  –Interference: more difficult to recall an item if there are other similar items (activated chunks) in memory

Discrimination principle

Difficulty of retrieval determined by candidates that exist in memory relative to retrieval cues

Long-Term Memory

There are two types of long-term memory: *episodic memory* and *semantic memory*.

- **Episodic memor**y represents our memory of events and experiences in a serial form. It is from this memory that we can reconstruct the actual events that took place at a given point in our lives.

- **Semantic memory**, on the other hand, is a structured record of facts, concepts and skills that we have acquired. The information in semantic memory is derived from that in our episodic memory, such that we can learn new facts or concepts from our experiences.

- Semantic memory is structured in some way to allow access to information, representation of relationships between pieces of information, and inference. One model for the way in which semantic memory is structured is as a network. Items are associated to each other in classes, and may inherit attributes from parent classes. This model is known as a *semantic network*. As an example, our knowledge about dogs may be stored in a network such as that shown in Figure.

- Holds mass of knowledge; facts, procedures, history

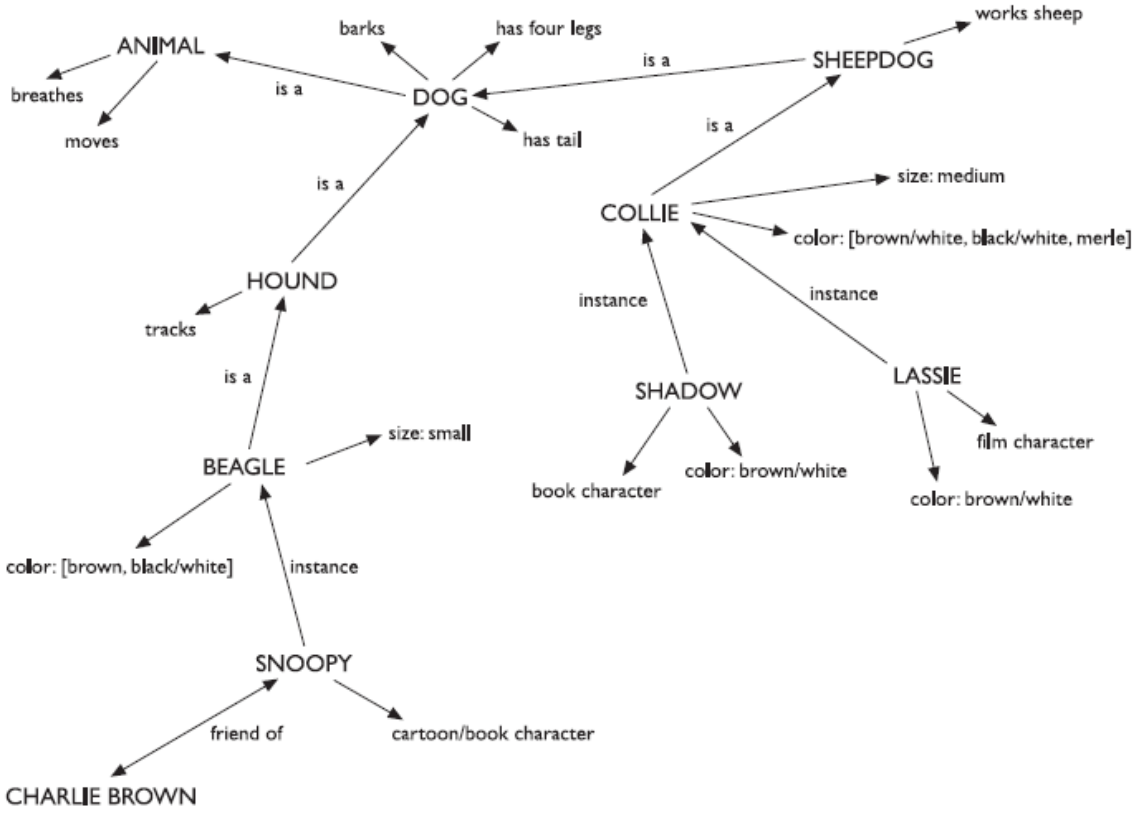
Two types

–Procedural –IF-THEN rules

–Declarative –facts

- Declarative memory consists of a network of related chunks where edge in the network is an association (semantic network)

LTM -Semantic Network Example



- Fast read, slow write
- Infinite storage capacity, but you may forget because:

  –Cannot find effective retrieval cues

  –Similar associations to other chunks interfere with retrieval of the target chunk (discrimination principle)

Cognitive Processor

- Implements "cognition" Operation called cognitive/production/decision cycles -A pattern matching process.

  –IF side tests for a particular pattern in declarative memory

  –When IF side matches, THEN side is executed (called rule firing)
- A cycle completes when no more firing is possible
- Activate motor component (ACT)

  –Fire another rule

  –Change WM/declarative memory (thus helping in other cycles)

Cognitive Processor Principle

Principle of recognize-act cycle

–Recognize: activate associatively-linked chunks in LTM
–Act: modify contents of WM

- Cycle time = ~70ms
- Cognitive System Principles

Uncertainty principle: Decision time increases with the uncertainty about the judgment to be made, requires more cognitive cycles

Variable rate principle: Cycle time is shorter when greater effort is induced by increased task demands or information loads; it also diminishes with practice

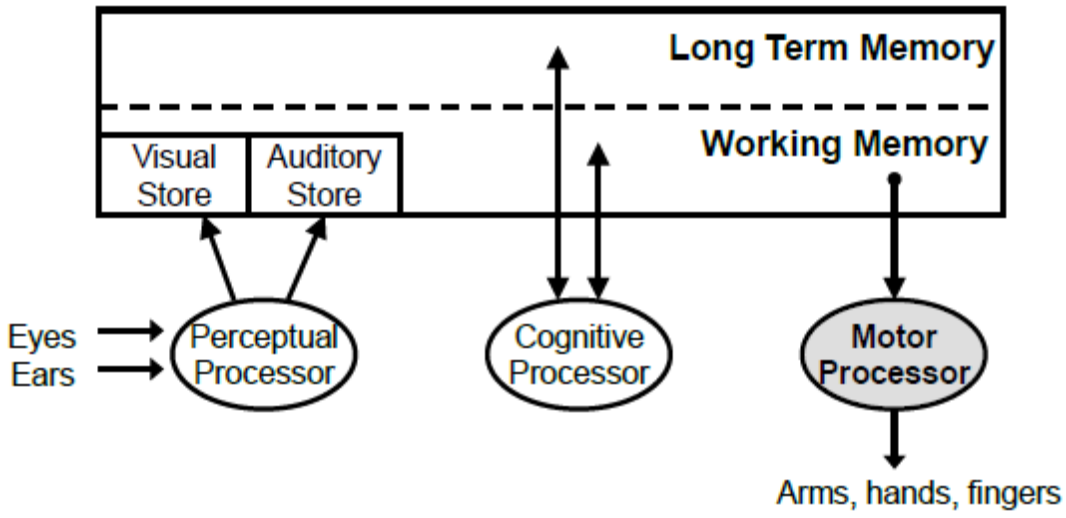
Power law of practice

$$T_n = T_1 * n^{-\alpha}$$

Here, T_n is the task completion time at the n^{th} trial, T1is the task completion time in the first attempt and αis learning constant (usually taken as 0.4)

Motor System

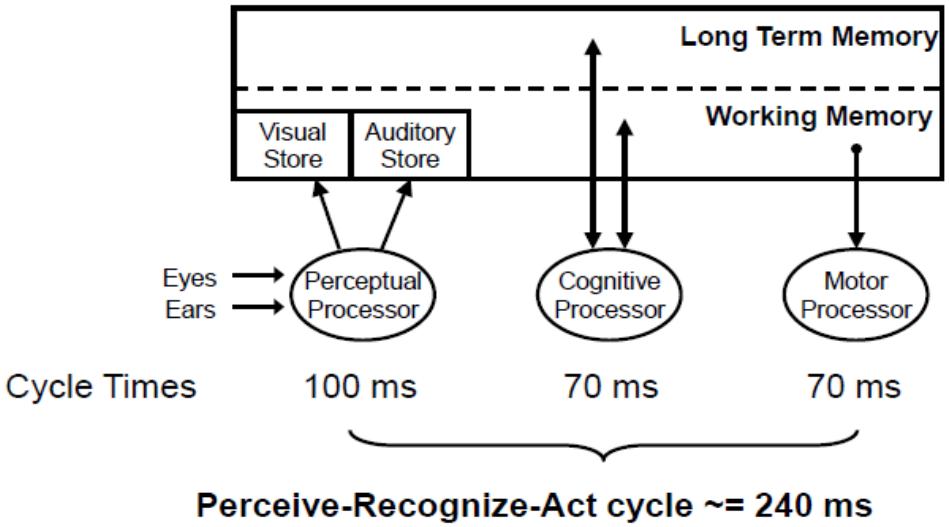
- Translates thoughts into actions
  –Head-neck and arm-hand-finger actions



Motor Processor

- Controls movements of body
  –Movement composed of discrete micro-movements
  –Micro-movement lasts about 70ms
  –Cycle time of motor processor is about 70ms
- Principle: Fitts' law

Putting It All Together



Principles –Summary

- Basis of the model
  –P0: Recognize-Act cycle of the cognitive processor
  –P8: Rationality principle
  –P9: Problem space principle
- Other 7 principles tend to describe ways of estimating duration of operators;
  –P1 --Variable perceptual processor rate
  –P2 --Encoding specificity principle
  –P3 --Discrimination principle
  –P4 --Variable cognitive processor rate principle
  –P5 --Fitts's law
  –P6 --Power law of practice
  –P7 --Uncertainty principle

Example 1

A user sits before a computer terminal. Whenever a symbol appears, s/he must press the space bar. What is the time between stimulus and response?

Tp (perceive the symbol) + Tc (recognize the symbol) + Tm (press key) = 240 ms
Tp = perceptual cycle time
Tc = cognitive cycle time
Tm = motor cycle time

Example 2

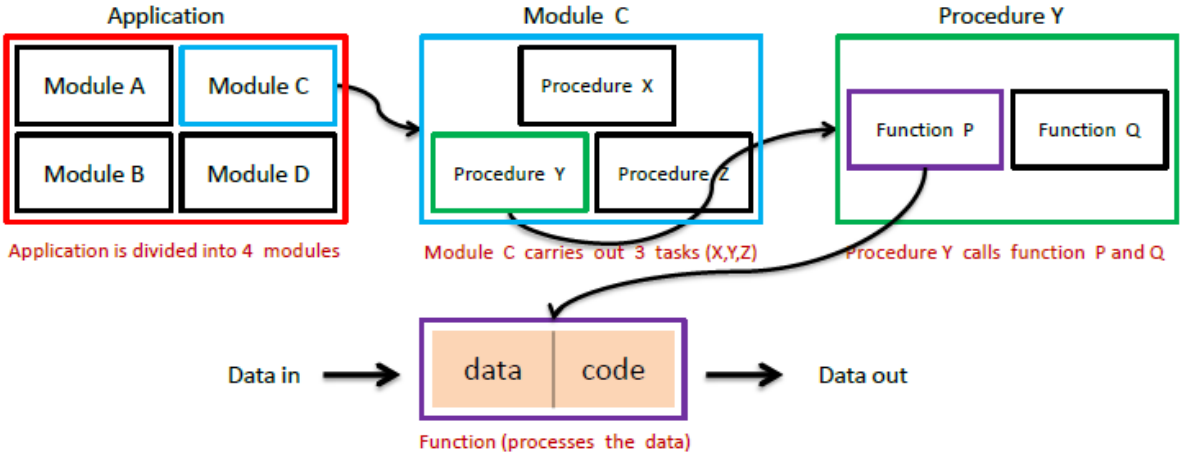
Two symbols appear on the computer terminal. If the second symbol matches the first, the user presses "Y" and presses "N" otherwise. What is the time between the second signal and response?

Tp + 2Tc (compare + decide) + Tm = 310 ms

Object Oriented Programming

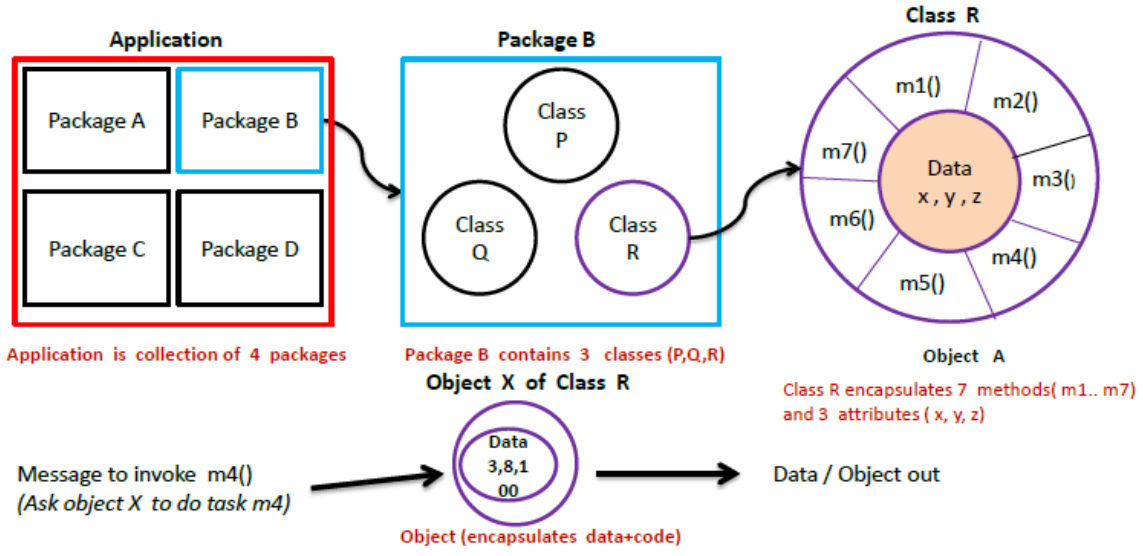
Procedural Programming Paradigm (PPP)

- In this paradigm (functional paradigm) the programming unit is a function.
- Program is divided into self-contained and maintainable parts called 'modules' (e.g. A,B,C,D).
- A module contains procedures -self-contained code that carries out a single task (e.g. X,Y,Z).
- A function (e.g. P, Q) is a self-contained code returning value to the calling procedure (e.g. Y).
- In PPP all computations are done by applying functions.
- Procedural paradigm separates data of program from instructions that manipulate the data.



Object Oriented Programming Paradigm (OOPP)

- Paradigm describes a system as it exists in real life based on interactions among real objects.
- Application is modeled as collection of related objects that interact and do your work (task).
- The programming unit is a class representing collection of similar real world objects.
- Programming starts with abstraction of useful real world objects and classes.
- Application is divided into multiple packages (e.g. A, B, C, D).
- A package is a collection of classes (e.g. P, Q, R) fulfilling group of tasks or functionality.
- A class is an encapsulated (data + code) definition of collection of similar real world objects.
- OOPP binds together data and the instructions that manipulate the data into a class.



Objects

- Objects are key to understanding object-oriented technology. Look around right now and you'll find many examples of real-world objects

DOG

State	Name Color Breed Hungry
Behavior	Barking Fetching Wagging Tail

BICYCLE

State	Current Gear Current Speed Current Pedal Cadence (rhythm)
Behavior	Changing gear Changing pedal cadence Applying brakes

- Real-world objects share two characteristics: They all have state and behavior
- Try and identify the State & behavior of the two objects Dog and Bicycle.

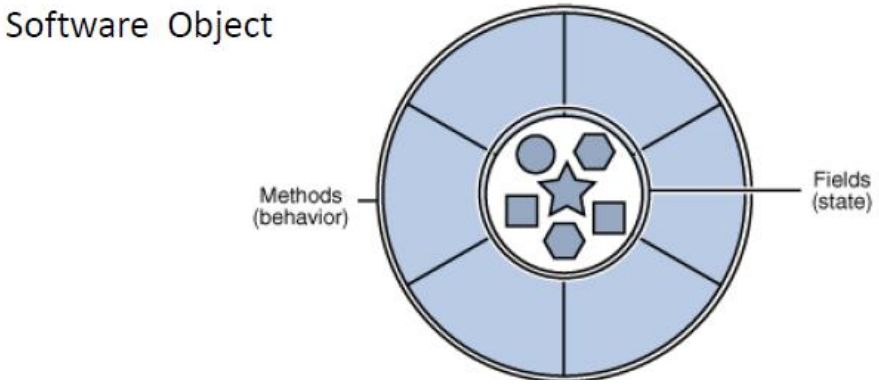
TABLE LAMP

State	On/Off
Behavior	Turning On Turning Off

DVD PLAYER

State	On /Off Current Volume Current Station
Behavior	Turn on Turn off Increase volume Decrease volume, Seek Scan Tune

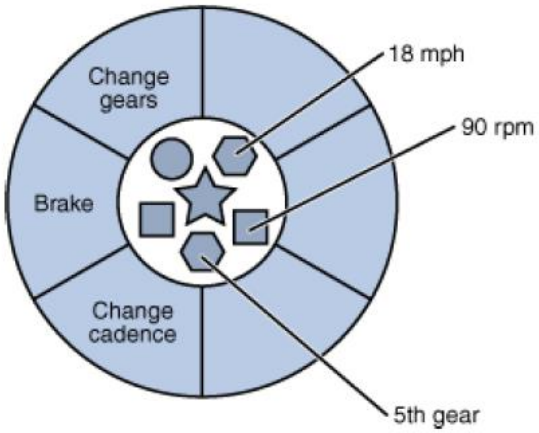
Software Objects -fields and methods



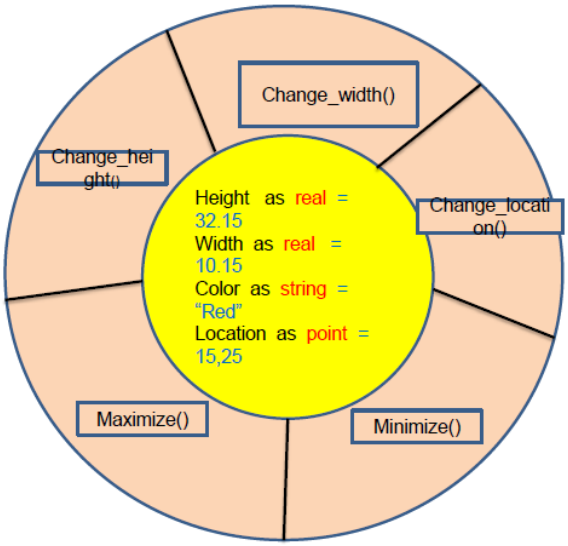
- An object stores its state / information in fields (attributes) and exposes its behavior through methods (member functions)

Objects -Data Encapsulation

- Objects methods can only change object's internal state.
- Hiding this internal state & requiring all interaction to be performed through an object's methods is known as data encapsulation.

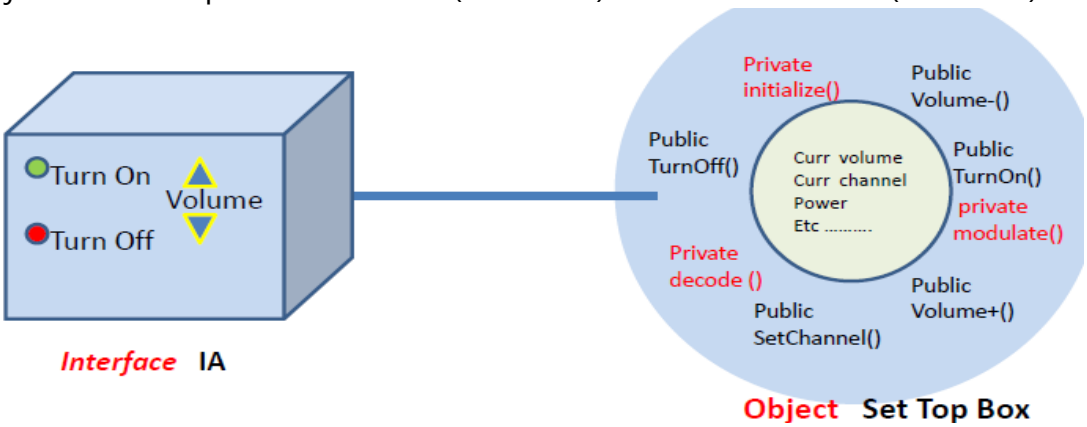


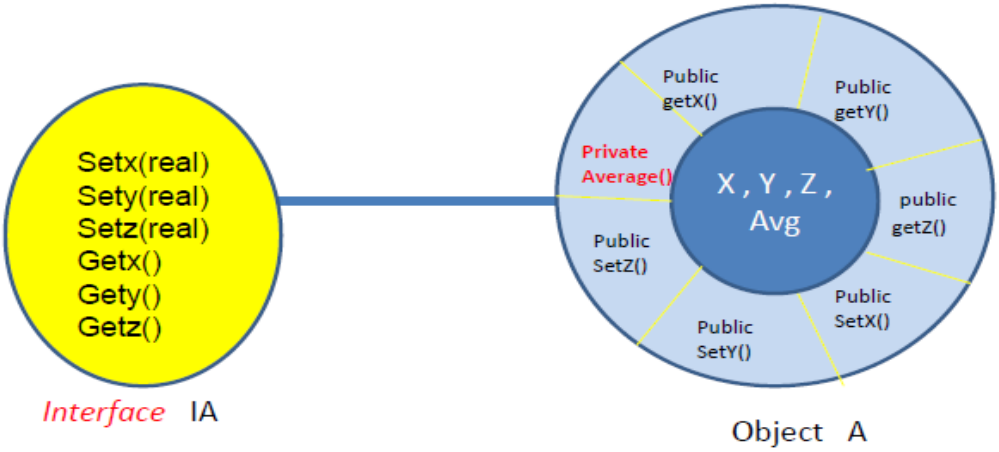
Window Object



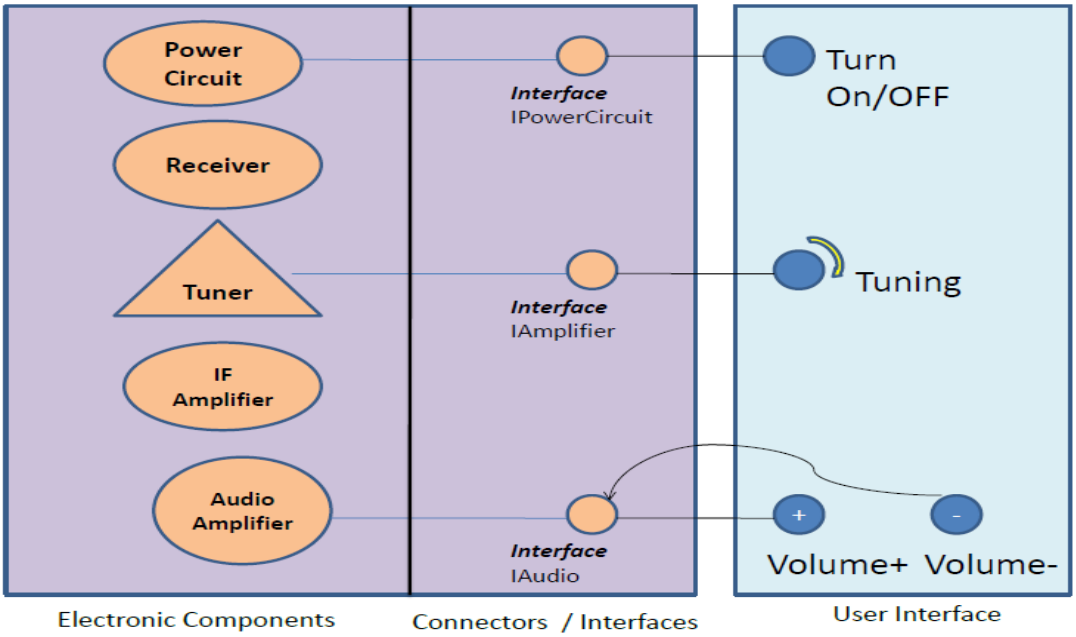
Objects -Public Interface

- Other objects can change the state of an object by using only those methods that are exposed to the outer world through a public interface. This help in data security.
- Object is an encapsulation of data (attributes) as well as methods (functions)





Object-based application



Benefits of object-based application development

Modularity:

- Source code for an object can be maintained independently in a class. Once created, an object can be easily passed around inside the system.
- E.g. Class Maths, Class string, Class Window etc.

Information-hiding:

- By interacting only with an object's methods, the details of its internal implementation remain hidden from the outside world. E.g. Interface IMaths, Interface IStringInterfaceIWindow

Code re-use:

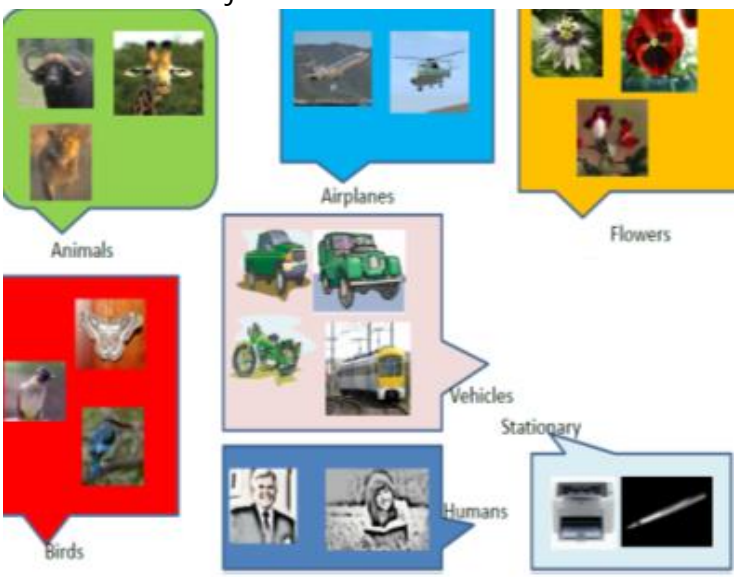
- If an object already exists (perhaps written by another software developer), you can use that object in your program. This allows specialists to implement/test/debug complex, task-specific objects, which you can then trust to run in your own code.
- e.g. Standard Classes available packages which can reused using their interfaces.

Pluggability and debugging ease:

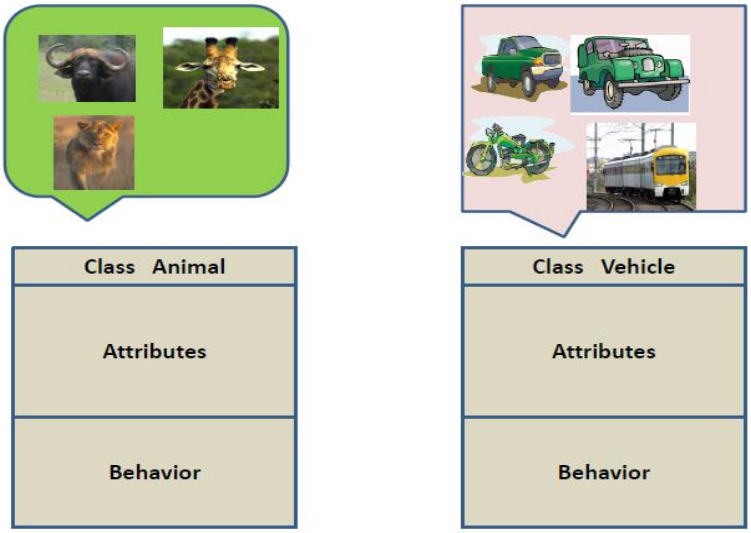
- If a particular object turns out to be problematic, you can simply remove it from your application and plug in a different object as its replacement. This is analogous to fixing mechanical problems in the real world. If a bolt breaks, you replace it, not the entire machine.

Class:

- Collection of objects that share common attributes and behavior (methods)

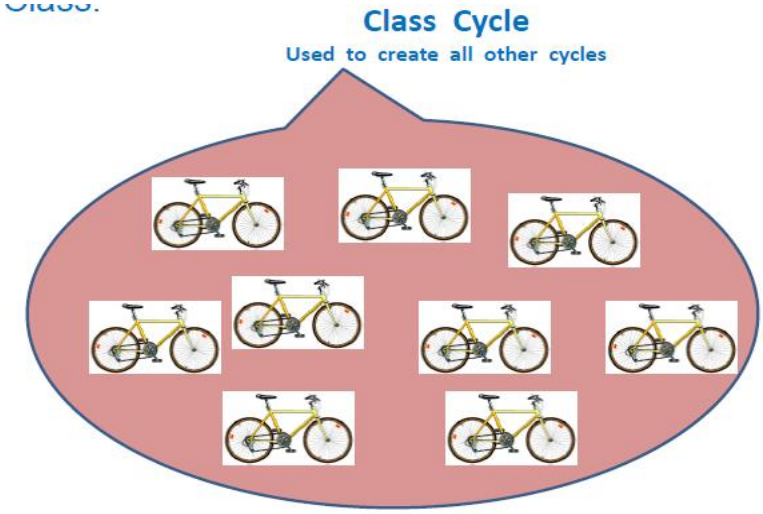


Class –Structure



Class -Blueprint of similar object

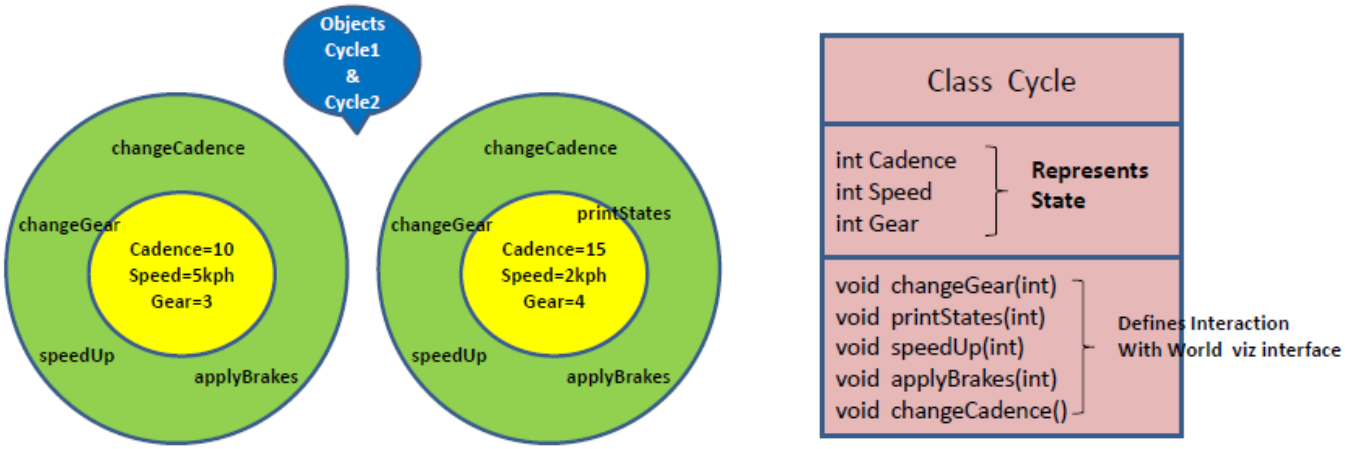
- Factory of cycles of same make and model with same features and same components. Built from same blueprint. This blueprint is called a Class.
- In object-oriented terms, we say that your cycle is an instance of the class of objects known as cycles. A classis the blueprint from which individual objects are created. The attributes and operations defined by a class are for its objects, not for itself. A class is a logical construct, an object has physical reality.



Class Definition –Example

- Classes are passive and which do not communicate but are used to create (instantiate) objects which interact.
- The responsibility of creating and using new Cycle objects belongs to some other class in your application.

```
class Cycle
 {
 int cadence = 0;
 Int speed = 0;
 Int gear = 1;
 void changeCadence(int newValue)
 { cadence = newValue; }
 void changeGear(intnewValue)
 { gear = newValue; }
 void speedUp(intincrement)
 { speed = speed + increment; }
 void applyBrakes(intdecrement)
 { speed = speed -decrement; }
 void printStates()
 { System.out.println("cadence:"+cadence+"
 speed:"+speed+" gear:"+gear); }
 }
```



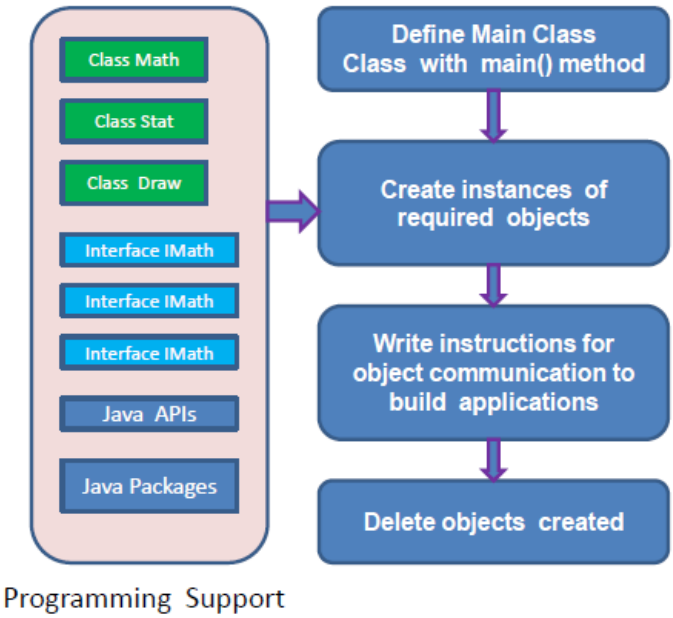
Instantiating objects from class

Cycle Democlass that creates two separate cycle objects and invokes their methods:

```
Class CycleDemo
{
public static void main(String[] args)
{ // Create two different Bicycle objects
Bicycle bike1 = new Bicycle();
Bicycle bike2 = new Bicycle();
// Invoke methods on those objects
bike1.changeCadence(50);
bike1.speedUp(10);
bike1.changeGear(2);
bike1.printStates();
bike2.changeCadence(50);
bike2.speedUp(10);
bike2.changeGear(2);
bike2.changeCadence(40);
bike2.speedUp(10);
bike2.changeGear(3);
bike2.printStates(); } }
```

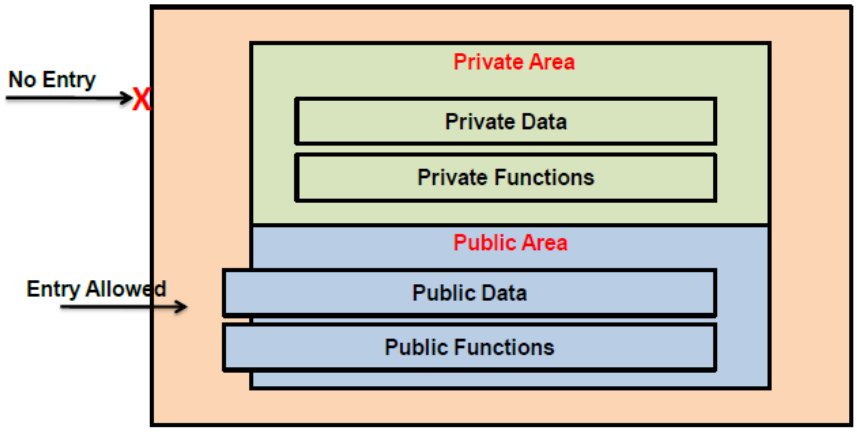
Main Class -Is a class that contains one method, called main. The main method contains a series of instructions to create objects and to tell those objects to do things.

Setting an Application's Entry Point



Object-Oriented Principle –Encapsulation

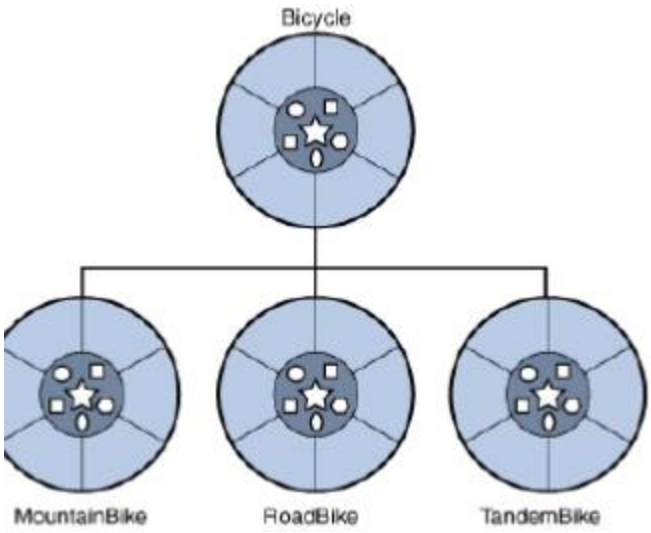
- Encapsulation is the mechanism that binds together the code and the data it manipulates, and keeps both safe from outside interference and misuse



Object-Oriented Principle –Inheritance

- Inheritance is the process by which one object acquires the properties of another object. By use of inheritance, an object need only define all of its characteristics that make it unique within its class, it can inherit its general attributes from its parent. This will be clear from the following example
- Mountain bikes, road bikes, and tandem bikes all share the characteristics of bicycles (current speed, current pedal cadence, and current gear).

- Yet each has some features that makes them different: tandem bicycles have two seats and two sets of handlebars; road bikes have drop handlebars; some mountain bikes have an additional chain ring, giving them a lower gear ratio.
- Object-oriented programming allows classes to inherit commonly used state and behavior from other classes.
- In this example, Bicycle now becomes the super class of Mountain Bike, Road Bike, and Tandem Bike.

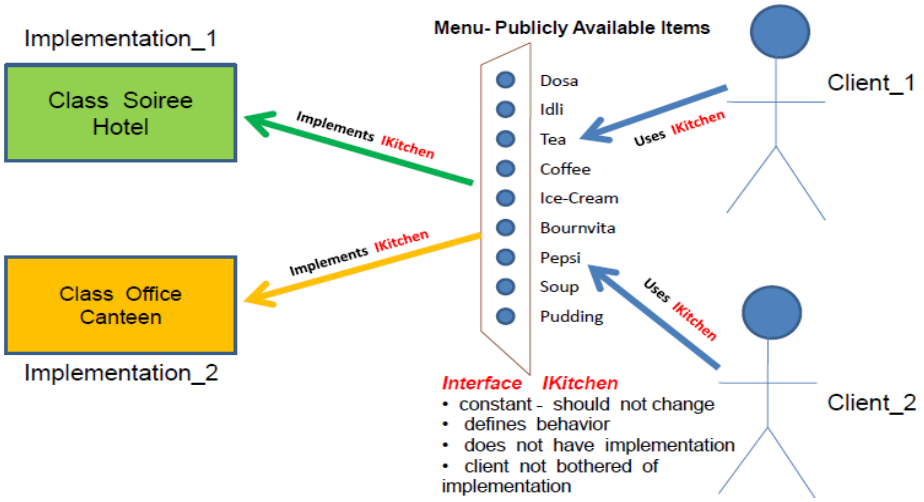


Creating subclass -at beginning of class use the extends keyword, followed by name of the class to inherit from:

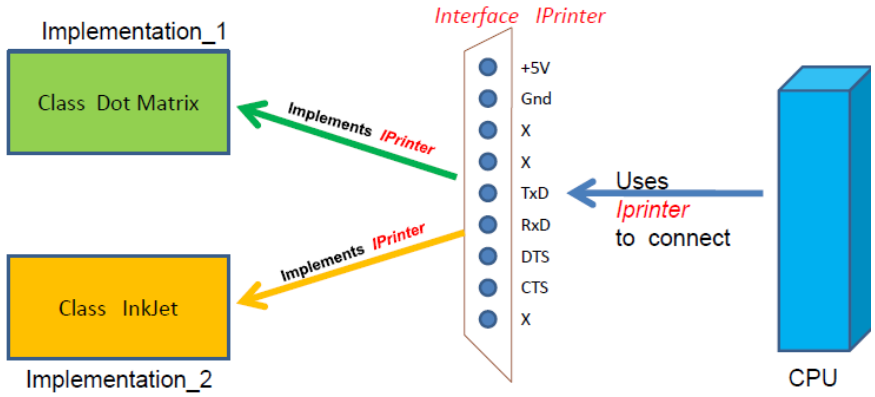
```
class MountainBike extends Bicycle
{
// new fields and methods defining a mountain bike would go here
}
```

Interfaces

- Public methods form the object's interface with the outside world
- Interface is a group of related methods with empty bodies (pure virtual functions)



- Public methods form the object's interface with the outside world
- Interface is a group of related methods with empty bodies (pure virtual functions)

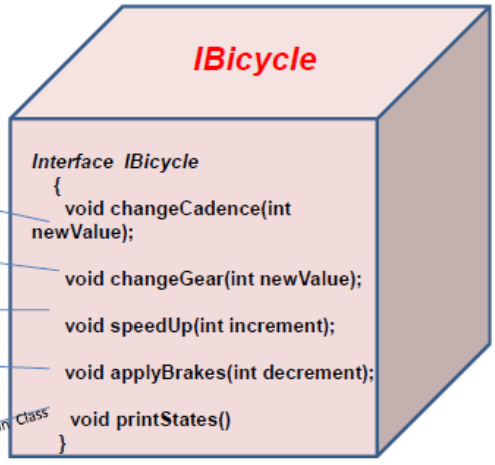


Class as implementation of interface

A bicycle's behavior, if specified as an interface, might appear as follows

e.g. Class Soiree **implements** *IKitchen*

```
Class Bicycle implements IBicycle
{
        int cadence = 0;
        int speed = 0;
        int gear = 1;

        void changeCadence(int newValue)
            { cadence = newValue; }

        void changeGear(int newValue)
            { gear = newValue; }

        void speedUp(int increment)
            { speed = speed + increment; }

        void applyBrakes(int decrement)
            { speed = speed - decrement; }

        void printStates()
            {
System.out.println("cadence:"+cadence+"
            speed:"+speed+" gear:"+gear); }

}
```



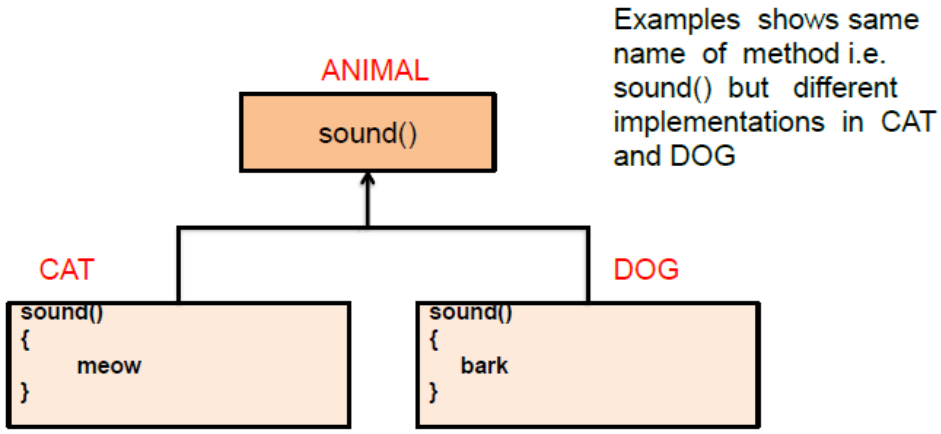
To implement interface, the name of your class would change (to a particular brand of bicycle, for example, such as ACMEBicycle), and you'd use the implements keyword in the class declaration

Interfaces define an application

- An interface makes a class formal about the behavior it promises to provide.
- Interfaces form a contract between the class and the outside world and this contract is enforced at build time by the compiler.
- If your class claims to implement an interface, all methods defined by that interface must appear in its source code before the class will successfully compile.

Object-Oriented Principle –Polymorphism

- Polymorphism (from Greek, meaning 'many forms') is a feature that allows same interface (method name) to be used for a multiple class of actions depending on context.



Packages and API

- A package is a namespace that organizes a set of related classes and interfaces like a folder
- Software written in the Java programming language can be composed of hundreds or thousands of reusable classes, it makes sense to keep things organized by placing related classes and interfaces into packages.
- Java Platform provides Class library which has huge number of classes organized in packages which is also called API.

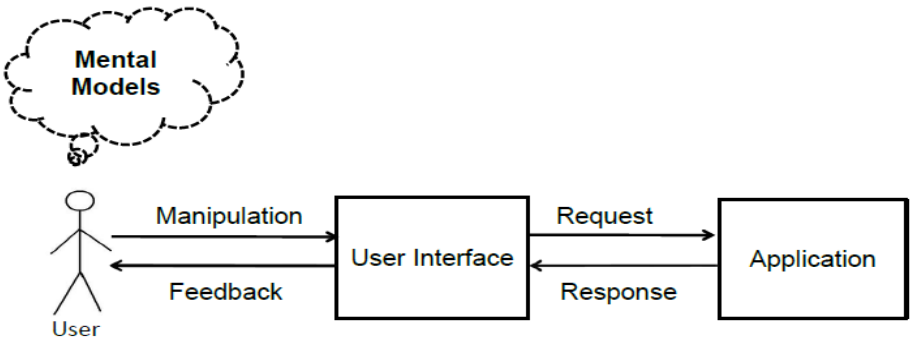
Object Oriented Modeling of User Interface Design

Introduction

- Presently user interfaces are built using rapid UI builders with extensive use of graphical element libraries
- This ease in deploying an UI may lead to haphazard designs
- To design usable interfaces we need to specify interaction goals
- Formal specification of UI leads to robust and effective UI
- Object orientation allows reuse of existing and tested UI elements
- Object oriented specification of UI should be directed towards implementer and the user

Interaction

- End user relies on mental models to predict behavior of the system.
- Mental model is the internal representation of the structure and behavior of the system or reality
- Accurate mental models helps user to understand and operate system efficiently
- Immediate and perceivable visual feedback of every interaction with interface helps end user to tune his mental model
- Perceptions about his interaction can further reinforce mental models towards ideal state.
- Thus the visible user interface should reflect user's mental model
- An object oriented application is a collection of software objects representing the application domain.
- Object –oriented interface therefore connects user with the real world manipulating software objects of the background application.

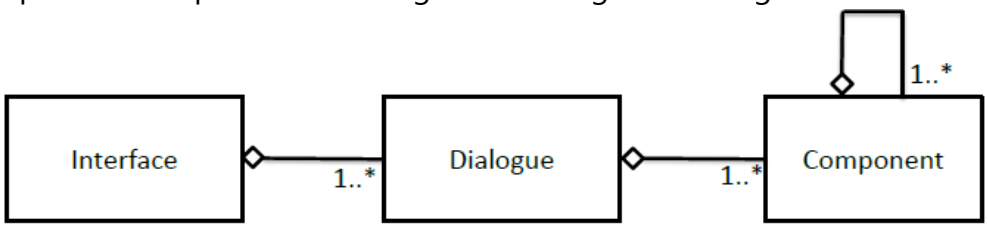


- Interface design should strive to make user successfully accomplish their goals with the help of interaction tasks and manipulation of required interface elements

Goals → Tasks → Interface elements

Interface Objects

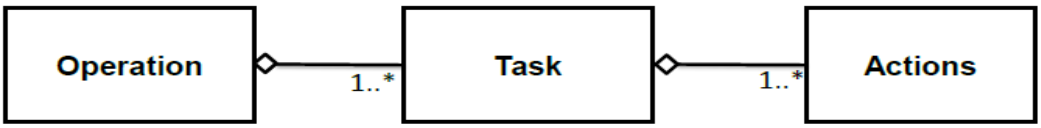
- At top level an interface is a collection dialogues.
- A dialogue is an aggregation of one or more components
- A component is further a collection of window elements that allows user to perform a meaning full action.
- Component is implemented using class having windowing elements or components



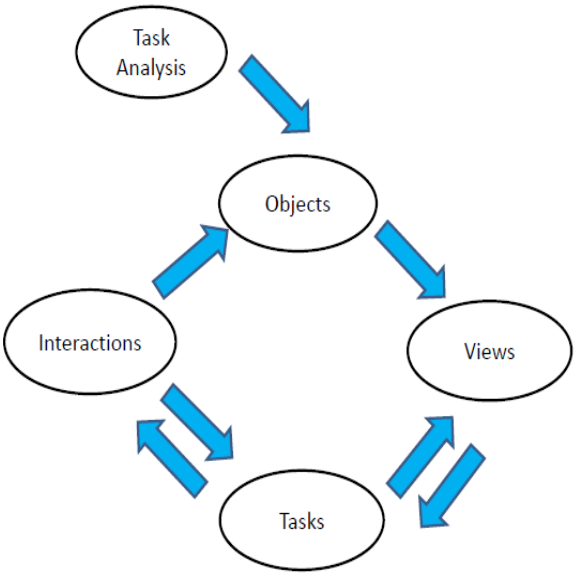
- Above figure shows a UML representation of an object oriented interface
- Filled diamond shows composition while empty diamond shows aggregation
- 1..* represents one or many multiplicity of relationship

Interface Actions

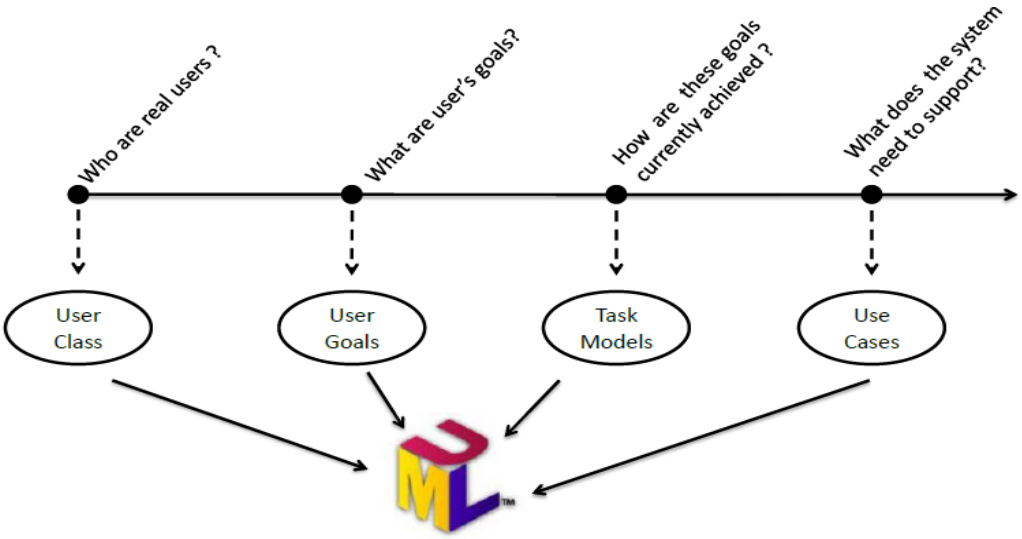
- User performs operation on interface to fulfill his goal by using system function.
- An USE CASE is a specific way of using system's functionality.
- A USE CASE specifies sequence of operations with interface to be performed.
- An operation is an activity for which the system is implemented. e.g. place order , draw picture etc .
- Operation consists of one or more tasks e.g. saving, drawing, deleting etc.
- An action is the smallest activity user performs on the user interface e.g. press button, drag picture etc.



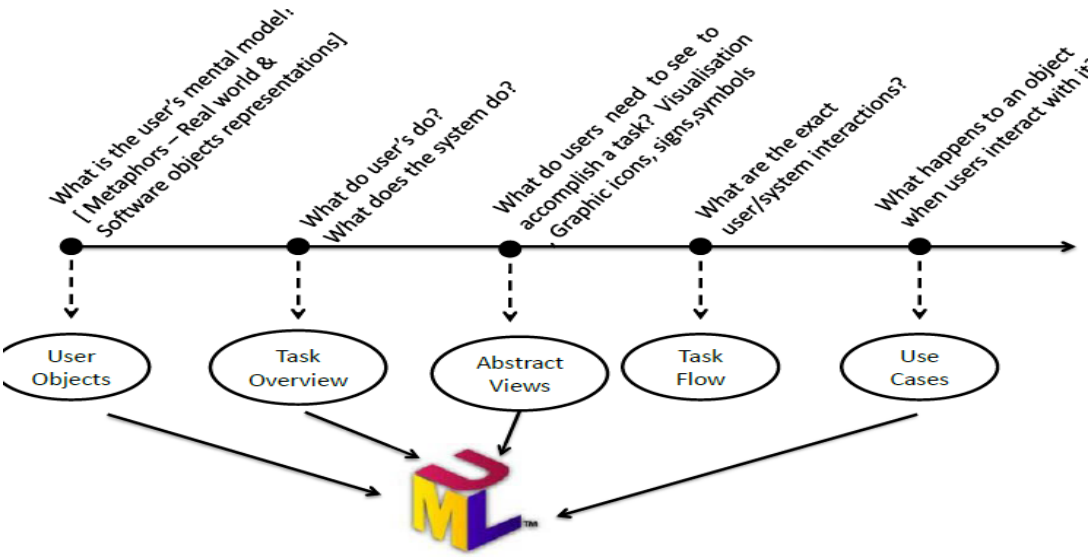
Methodology -Object Oriented UI design



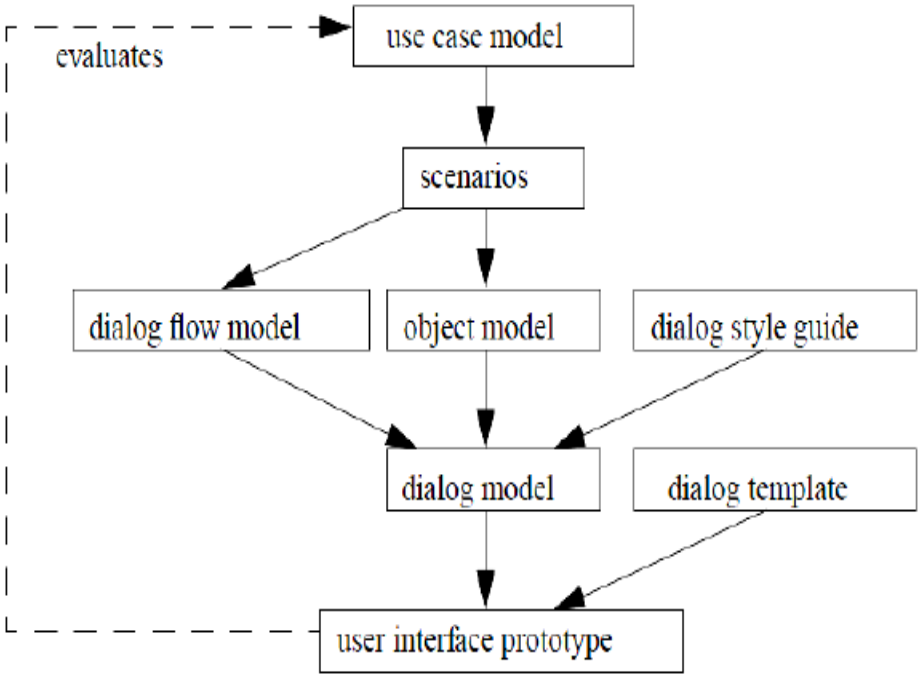
Methodology -Discovery Phase



Methodology -Abstract Design Phase



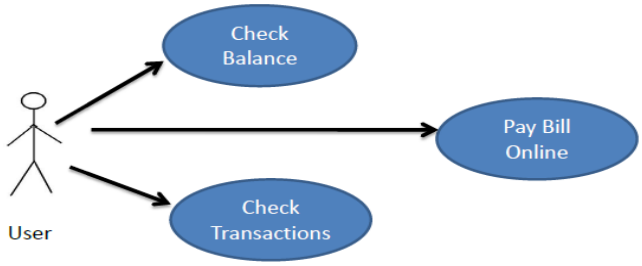
Methodology –Flow



Object-Oriented Modeling of User Interfaces

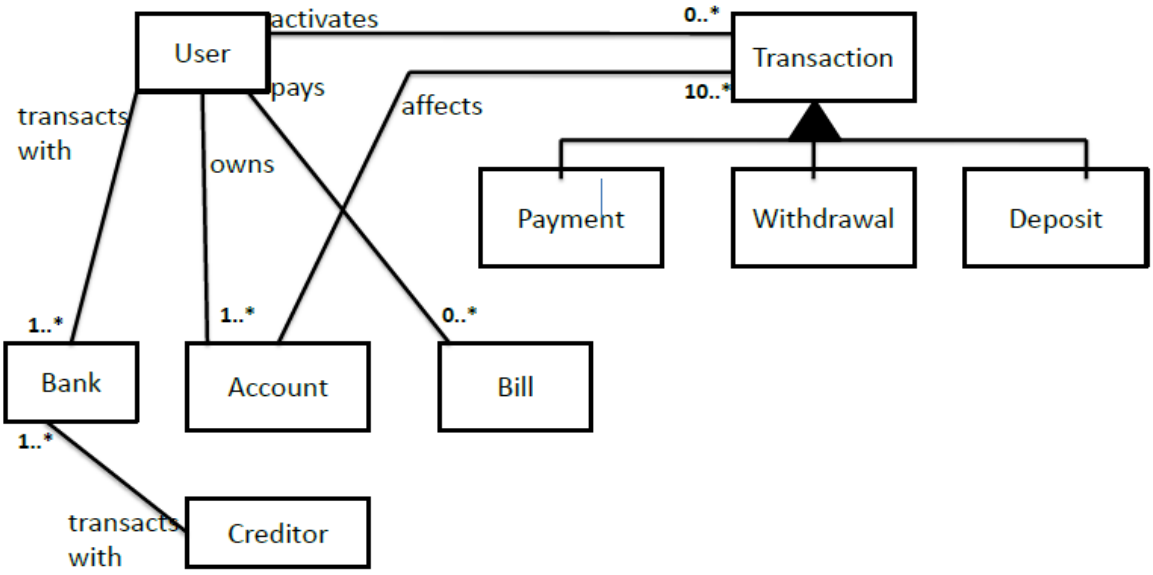
First step is analysis of user requirements –identifying USE CASES

- Design specifies the structure and components required each dialogue to accomplish the USE CASE
- Interfaces are then developed iteratively and tested against with USECASE specifications and various criteria viz. performance, usability etc. Example: Personal banking application.
- Why would a user interact with such application? What are his goals?
- USE CASES for personal banking application are



Object Analysis Model

- The sequence of operations documented for each USE CASE are analyzed to identify key objects in the application domain resulting into an object model.
- Key objects are called analysis objects and the diagram showing relationships between these objects is called object diagram.



Task (Operation) Analysis

- The sequence of tasks (user interactions and system feedback) are specified for each USE CASE which is called scenario.
- Scenario specifies detailed interaction with interface elements and feedback received.
- Operation or scenarios are analyzed to identify interaction tasks.

Examples

Scenario 1 : USE CASE (Operation) → Check Balance

1. Select account
2. View balance

Scenario 2 : USE CASE (Operation) → Pay Bill Online

1. Select account
2. Update bill
3. Put bill for payment

Scenario 3 : USE CASE (Operation) → Check Transactions

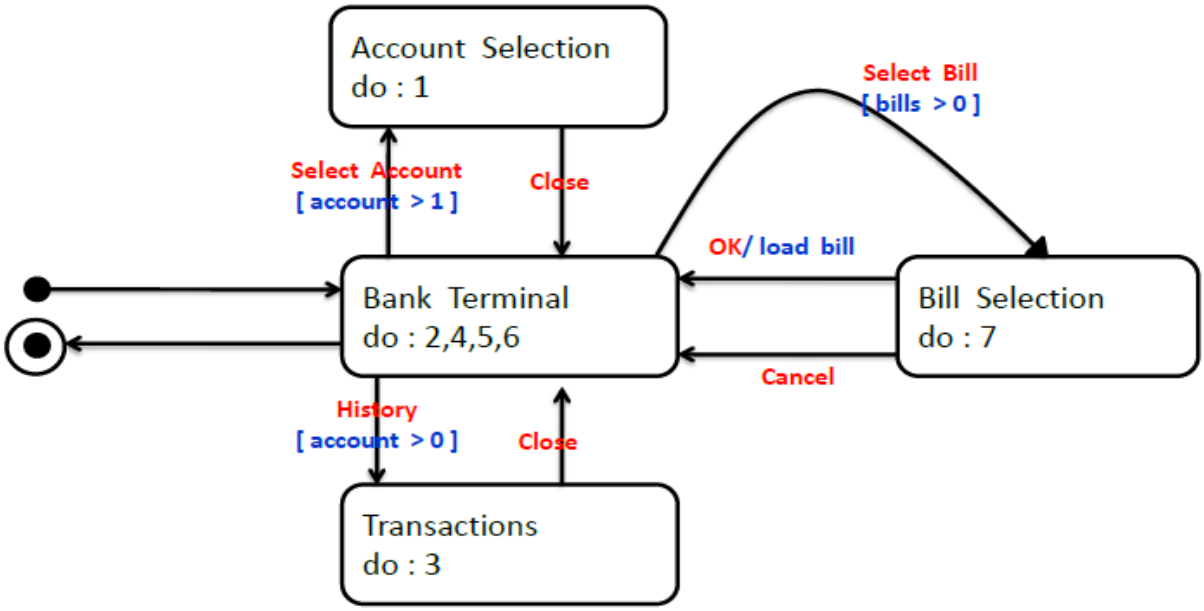
1. Select account
2. View transactions

Task List

- Select account
- View balance
- View transactions
- Update bill
- Put bill for payment

Behavioral Structure Specification -dialogues transitions

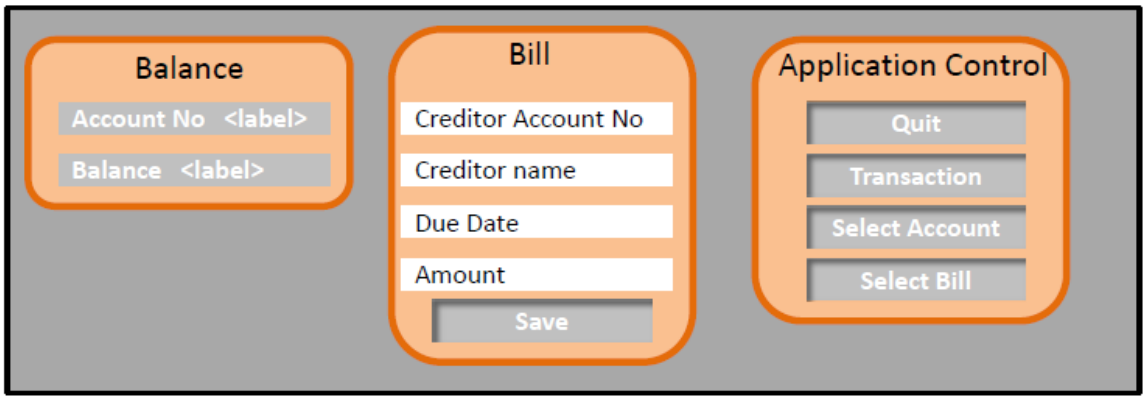
- State transition diagram shows relationship between dialogues (windows) of an interface
- Every dialogue is considered as a state of the system.
- Events (user actions) that trigger transition from one dialogue to another are specified
- A do statement within dialogue specifies task number from the scenario.
- Conditions for transition from one dialogue to another are written in square [] brackets
- Operation during transition are written after trigger a preceded with slash /.



Interface Components Specification

- Specifies UI components required to carry out task within each dialogue
- Types of UI components → input (push button, slider), output (label, image)
- Show below is dialogue: Bank Terminal having instances of 3 components

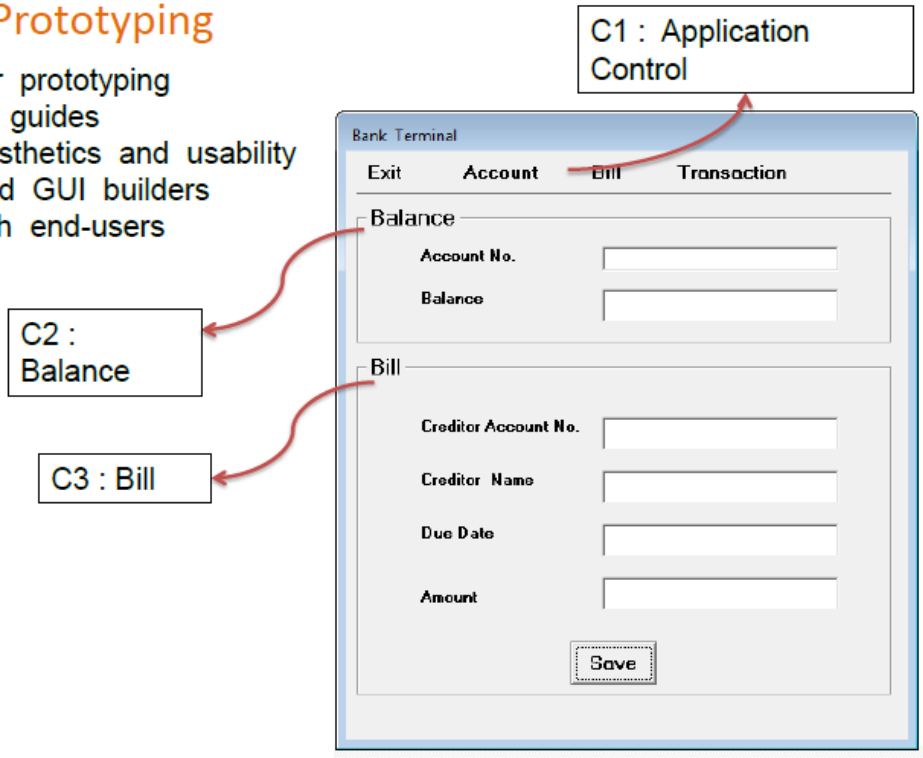
Bank Terminal Dialogue



- Structural elements: window, menu, icons, controls (widgets), tabs ….
- Interactional elements: cursor, pointer, selection handle ….
- Component increases reusability and consistency of UI
- A component can be created using elements from scratch or from available standard dialogue classes or components from libraries (packages).

Interface Prototyping

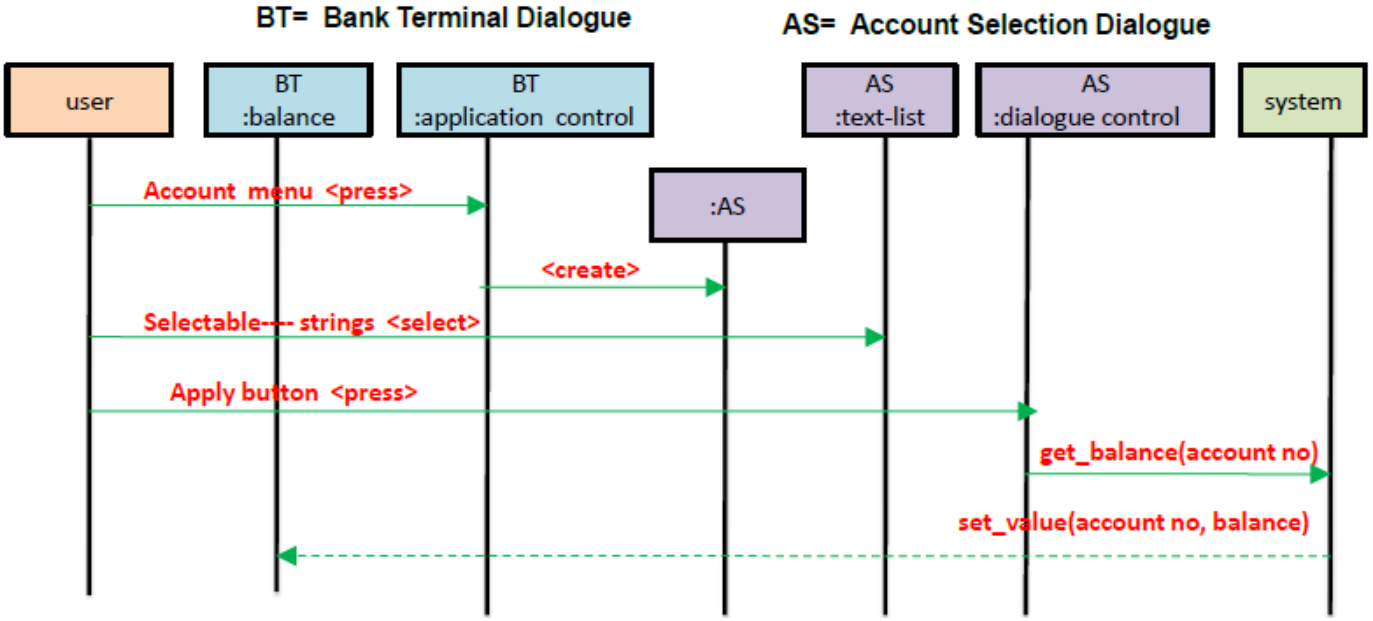
Guidelines for prototyping
- Follow style guides
- Consider aesthetics and usability
- Use standard GUI builders
- Validate with end-users



Thorough designing of dialogues and components leads to identification of highly reusable interface elements which finally leads to harmonized user interfaces

Task Specification and Flow

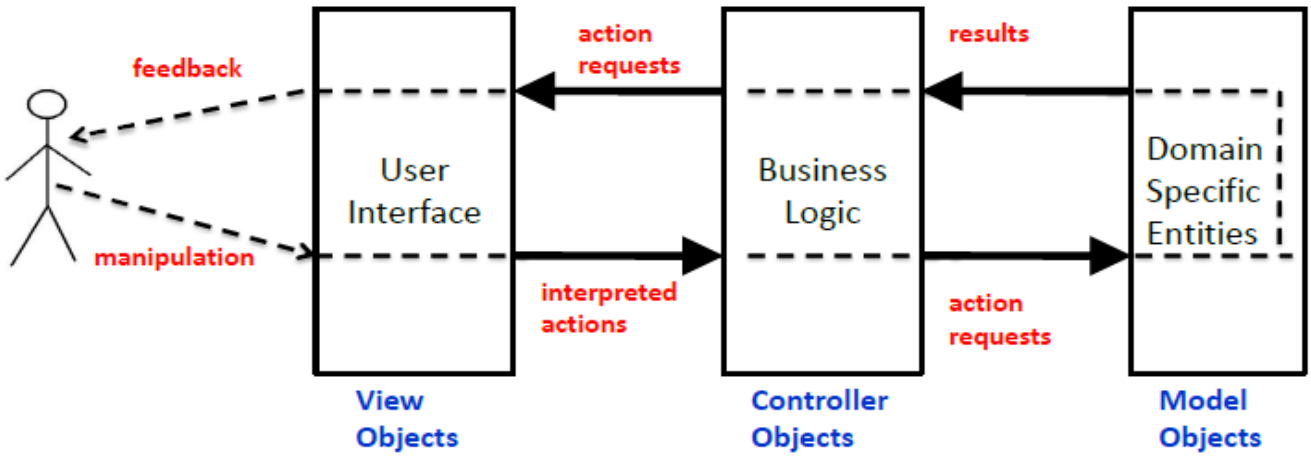
- Specifies sequence of actions to be performed on final user interface for a task.
- Tasks are selected from use case scenarios.
- Designers performs a walk-through with final interface for each task.
- Specification of event trace of interactions among user, interface and system can also be depicted with interaction diagram as below for "selecting an account".



- Objects shown as rectangles at top, object interaction shown by arrows and messages
- Tail of the arrow shows the invoking object and time axis runs from top to bottom

Converting UI Specification into Language Specific Design

- Each application consists of three types of objects
- Domain or model objects represent the real world (also called entity objects)
- Model objects are controlled by controller objects and are unaware of view objects.
- View objects are on boundary of system interacting with end user.
- View objects only knows how to present feedback and accept user inputs. They do not take decisions or process information.
- The controller objects receive the request from view objects which makes application specific decision through model objects. This is called MVC architecture as shown below



- Every dialogue in analysis model forms view object in design model e.g Balance (View), Bill (View), Application Control (View)
- Dialogue components become object members of respective view object.
- There is a single controller object for every view object
- Single controller may be related to more than one model object and vice versa.
- Manipulation (set) and feedback (get) behavior of component is implemented as member functions of the view component class. Query methods are designed for controller object.
- Application specific c logic is implemented in controller classes.
- Model classes have domain specific knowledge.
- Event traces and state machines in analysis phase are used to find member function of design classes.
- Design model in the next diagram depicts all classes of the application

Fig showing all classes in the Design Model