CS242
Advanced Programming Concepts in Java
Mobile HCI
12-02-2013
Mobile HCI
More on creating objects: static factory method

Read: Effective Java 2nd Edition, Chapter 2
“Creating and Destroying Objects”

PR#2: due Wednesday, 12/04
Extra Credit: due Friday, 12/06
Final Exam: Monday, 12/09, 7 – 10 pm, SC162
Principles of Visual Interface Design

- Use visual properties to group elements and create a clear hierarchy
- Provide visual structure and flow at each level of organization
- Use cohesive, consistent, and contextually appropriate imagery
- Integrate style and function comprehensively and purposefully
- Avoid visual noise and clutter
Website Composition

- Users view a website in an F shaped pattern
- Users won't read your text thoroughly; they prefer to skim
Mobile HCI

- What’s different about mobile HCI?
- MobileHCI 2013

- The 10 principles of Mobile Interface Design
Mobile is different

Limitations:
- small screen size
- limited memory
- intermittent connectivity
- low bandwidth
- limited battery

Sensors:
- location, movement, acceleration, orientation, proximity, etc.

Interaction:
- (soft) keyboard
- finger swipes
- stylus
- etc.
1. Mobile mindset

- be focused – more is not better
- be unique – what distinguishes your app?
- be charming – friendly, reliable and fun
- be considerate – take the user’s point of view
10 Principles of Mobile Interface Design

2. Mobile context

- **bored** – provide something engaging, but that can be resumed later (e.g. games, social, …)
- **busy** – e.g. racing through airport, so need to accomplish task quickly and reliably
- **lost** – unfamiliar surrounding, or familiar place but something is unknown; sketchy connectivity and battery life are big concerns, so provide off-line support if possible
3. Global guidelines

- responsiveness – absolutely a must!
- polish – it must look good!
- design for thumbs, rather than fingers, for touch interfaces (e.g. 44 pixels is thumb friendly for UI elements); placement of targets important
- content – touch allows direct interaction, so minimize the use of buttons, tab bars, check boxes, etc.
- add controls at the bottom of the screen (unlike webpages, where mouse interaction is typical)
- avoid scrolling
4. **Navigation Models**

- none – single screen utility apps (e.g. weather)
- tab bar – 3 to 6 content areas
- drill down – list and detail content hierarchy (like in settings)
5. **User Input**

- make typing easy
- select the keyboard that makes sense (text, number, etc.)
- auto-correct can be frustrating if not done well
- consider supporting landscape orientation
6. **Gestures**

- invisible to user, so make it easy to discover; how?
- multitouch vs. one-handed
7. Orientation

- portrait vs. landscape
- consider adding an orientation lock
8. Communication

- provide instant feedback (tactile, visual, sound)
- minimize use of modal alerts
- confirmation
10 Principles of Mobile Interface Design

9. Launching
   - if not the first time, resume operations where they were left off
   - present image when loading (not controls)
10 Principles of Mobile Interface Design

10. First Impressions matter!

Message - Mental impression you create through visual design.

Look and Feel - How the users will interact or interface with the app. (Think: The user will press this button because...)

Layout - Structural definition of the page including navigation, menus, tabs and lists. Different layouts for different devices?

Color - Psychology and Palette. (Color depth is a concern!)
More on creating objects

- Effective Java 2, **Item 1**: Consider **static factory methods instead of constructors**

- To allow a client to create an object, you can...
  - provide a public constructor
  - provide a public static factory method

- What is a **static factory method**?
  - a static method that returns an instance of a class
  - note: this is not the Factory Design pattern
String has several constructors, including:

public String()
- Initializes a newly created String object so that it represents an empty character sequence.

public String(byte[] bytes)
- Constructs a new String by decoding the specified array of bytes using the platform's default charset.

and so on...

Client code:  String name = new String(myinput);
String also has several static factory methods, including:

- static `valueOf(char[] data)`
  - Returns the string representation of the char array argument

- static `valueOf(boolean b)`
  - Returns the string representation of the boolean argument

Client code:

```java
String name = String.valueOf(mydata);
```
A class can provide its clients with static factory methods instead of, or in addition to, constructors.

Providing a static factory method instead of a public constructor has both advantages and disadvantages.
Advantages of static factory methods

One advantage of static factory methods is that, unlike constructors, they have names; for example:

- constructor for BigInteger which returns a BigInteger which is probably prime

```java
public BigInteger(int bitLen, int certainty, Random rnd)
```

client writes: `BigInteger bi = new BigInteger(100,80,r);`

- static factory method to do the same

```java
public static probablePrime(int bitLen, Random rnd)
```

client writes: `BigInteger.probablePrime`
A second advantage of static factory methods is that, unlike constructors, they are not required to create a new object each time they are invoked.

- So immutable classes can use precomputed instances, or to cache instances as they are constructed, and dispense them repeatedly to avoid creating unnecessary duplicate objects; e.g. `Boolean.valueOf(boolean)` never creates an object.

- This is essentially a Flyweight Design Pattern.

- Classes that do this are called “Instance-controlled”
A third advantage of static factory methods is that, unlike constructors, they can return an object of any subtype of their return type.

- this gives you flexibility in choosing the class of the returned object
- so, an API can return objects without making their class public (that is, hiding implementation classes)

Examples include the Collections Framework API and the Java Database Connectivity API (JDBC)

- the class java.util.EnumSet has no public constructors, only static factories; if a small enum, uses one method, if larger then uses another
Static Factory Methods

Example from the Java Collections Framework:

```java
public static <T> Collection<T> synchronizedCollection(Collection<T> c)
```

Returns a synchronized (thread-safe) collection backed by the specified collection. In order to guarantee serial access, it is critical that all access to the backing collection is accomplished through the returned collection.
Advantages of static factory methods

A fourth advantage of static factory methods is that they reduce the verbosity of creating parameterized type instances.

- for example, consider

```
Map<String, List<String>> m = new HashMap<String, List<String>>();
```

- with static factories, the compiler can infer the type parameters automatically (type inference); so the if the following static factory was in HashMap:

```
public static <K, V> HashMap<K, V> newInstance() {
    return new HashMap<K, V>();
}
```

the user can replace the above declaration with:

```
Map<String, List<String>> m = HashMap.newInstance();
```
Disadvantages of static factory methods

- The main disadvantage of providing only static factory methods is that classes without public or protected constructors cannot be subclassed.

- A second disadvantage of static factory methods is that they are not readily distinguishable from other static methods.
Common names for static factory methods

- `valueOf` – returns an instance that has the same “value” as its parameters (effectively a type-conversion method)
- `of` – a concise alternative to `valueOf` (used in `EnumSet`)
- `getInstance` – returns an instance that is described by the parameters but cannot be said to have the same value
- `newInstance` – like `getInstance`, except that it guarantees that each instance returned is distinct from all others
Common names for static factory methods, continued

- **getType** – like `getInstnace`, but used when the factory method is in a different class. Type indicates the type of object returned by the factory method

- **newType** – like `newInstance`, but used when the factory method is in a different class. Type indicates the type of object returned by the factory method
Effective Java 2, **Item2**: Consider a builder when faced with many constructor parameters

- note: static factories and constructors both have difficulties when there are a large number of optional parameters

- solution: **Builder Design Pattern**
  - cf. Head First Design Patterns, pp. 614–615