```java
package com.blindtigergames.werescrewed.entity.builders;

import java.util.HashMap;

import com.badlogic.gdx.graphics.Texture;
//... omitted
import com.blindtigergames.werescrewed.util.ArrayHash;

/**
 * EntityBuilder is meant to simplify creating entities and allow for extension
 * through inheritance and polymorphism. Will probably be a constant
 * work-in-progress as new Entity classes are added.
 *
 * I added this generic version of EntityBuilder to better allow for different
 * types of builders. Now new subclasses of EntityBuilder don't have to redefine
 * its parent's methods; you just have to specify the new type in the "extends"
 * tag, and the generic will handle the rest for you.
 *
 * @author Kevin
 *
 */
public class GenericEntityBuilder< B extends GenericEntityBuilder< ? >> {

    // Common to all builders
    protected String name;
    protected Vector2 pos; // in pixels
    protected float rot;
    protected Vector2 sca;
    protected IMover mover;
    protected boolean solid;
    protected String definition;
    protected ArrayHash< String, HashMap< String, String >> sounds;
    protected Array<String> soundlines;

    // Used for type+world construction
    protected EntityDef type;
    protected World world;

    // Used for texture+body construction
    protected Texture tex;
    protected Body body;

    public GenericEntityBuilder( ) {
        resetInternal( );
    }
```

```java
    protected void resetInternal( ) {
        name = "";
        pos = new Vector2( 0, 0 );
        rot = 0.0f;
        sca = new Vector2( 1, 1 );
        solid = true;
        mover = null;
        type = null;
        world = null;
        tex = null;
        body = null;
        sounds = new ArrayHash< String, HashMap< String, String >>( );
        soundlines = new Array<String>();
        definition = "";
    }

    // Simply resets the builder to initial state and returns it.
    @SuppressWarnings( "unchecked" )
    public B reset( ) {
        resetInternal( );
        return ( B ) this;
    }

    /**
     *
     * @param name
     *            - String name of entity, default is "noname"
     * @return EntityBuilder
     */
    @SuppressWarnings( "unchecked" )
    public B name( String n ) {
        name = n;
        return ( B ) this;
    }

    /**
     *
     * @param definition
     *            - String XML name of entity, default is "noname"
     * @return EntityBuilder
     */
    @SuppressWarnings( "unchecked" )
    public B definition( String d ) {
        definition = d;
        return ( B ) this;
    }

    /**
     *
     * @param def
     *            - EntityDef used to load body/texture information.
     * @return EntityBuilder
     */
    @SuppressWarnings( "unchecked" )
    public B type( EntityDef def ) {
        type = def;
        if ( type.getCategory( ) == EntityCategory.PLAYER ) {
            return ( B ) new PlayerBuilder( ).copy( this );
        }
        return ( B ) this.properties( def.getProperties( ) );
    }

    /**
     *
     * @param def
     *            - Runs the EntityDef function with the definition loaded from
     *            this name.
     * @return EntityBuilder
     */
    public B type( String def ) {
        return ( B ) type( EntityDef.getDefinition( def ) );
    }

    /**
     *
     * @param world
     *            - sets the current world of the created entity.
     * @return EntityBuilder
     */
    @SuppressWarnings( "unchecked" )
    public B world( World w ) {
        world = w;
        return ( B ) this;
    }

    /**
     *
     * @param body
     *            - sets the body of the created entity.
```

```
     * @return EntityBuilder
     */
    @SuppressWarnings( "unchecked" )
    public B body( Body b ) {
        body = b;
        world = b.getWorld( );
        return ( B ) this;
    }

    /**
     *
     * @param tex
     *            - sets the texture of the created entity.
     * @return EntityBuilder
     */
    @SuppressWarnings( "unchecked" )
    public B texture( Texture t ) {
        tex = t;
        return ( B ) this;
    }

    /**
     *
     * @param p
     *            - sets the position of the created entity in PIXELS.
     * @return EntityBuilder
     */
    @SuppressWarnings( "unchecked" )
    public B position( Vector2 p ) {
        return ( B ) positionX( p.x ).positionY( p.y );
    }

    /**
     *
     * @param x
     *            - new x position of the created entity (in pixels)
     * @param y
     *            - new y position of the created entity (in pixels)
     * @return EntityBuilder
     */
    @SuppressWarnings( "unchecked" )
    public B position( float x, float y ) {
        return ( B ) positionX( x ).positionY( y );
    }
```

```
        /**
         *
         * @param x
         *            - new x position of the created entity in PIXELS.
         * @return EntityBuilder
         */
        @SuppressWarnings( "unchecked" )
        public B positionX( float x ) {
            pos.x = x;
            return ( B ) this;
        }

        /**
         *
         * @param y
         *            - new y position of the created entity in PIXELS.
         * @return EntityBuilder
         */
        @SuppressWarnings( "unchecked" )
        public B positionY( float y ) {
            pos.y = y;
            return ( B ) this;
        }

        /**
         *
         * @param r
         *            - new angle of the created entity in radians
         * @return EntityBuilder
         */
        @SuppressWarnings( "unchecked" )
        public B rotation( float r ) {
            rot = r;
            return ( B ) this;
        }

        /**
         *
         * @param s
         *            - sets whether the created entity is solid or not.
         * @return EntityBuilder
         */
        @SuppressWarnings( "unchecked" )
        public B solid( boolean s ) {
            solid = s;
```

```
            return ( B ) this;
        }

        /**
         * Loads an entity's special properties from a hashmap.
         *
         * @param props
         *            - String/String hashmap containing the data
         * @return EntityBuilder
         */
        @SuppressWarnings( "unchecked" )
        public B properties( ArrayHash< String, String > props ) {
            if ( props.containsKey( "texture" ) ) {
                this.texture( WereScrewedGame.manager.get( props.get( "texture" ),
                        Texture.class ) );
            }
            // Handle sound tags
            boolean moreSounds = true;
            String tag;
            for (int i = -1; i < 99 && moreSounds; i++){
                if (i < 0){
                    tag = "sound";
                } else {
                    tag = "sound" + i;
                }
                if (props.containsKey( tag )){
                    for ( String line : props.getAll( tag ) ) {
                        soundlines.add( line );
                    }
                } else if (i >= 0){
                    moreSounds = false;
                }
            }
            return ( B ) this;
        }

        /**
         * Data-wise copy of another EntityBuilder into this one.
         *
         * @param that
         *            - the original builder to be copied.
         * @return EntityBuilder
         */
        @SuppressWarnings( "unchecked" )
        public B copy( GenericEntityBuilder< ? > that ) {
```

```java
        name = that.name;
        pos = that.pos;
        rot = that.rot;
        sca = that.sca;
        solid = that.solid;
        mover = that.mover;
        type = that.type;
        world = that.world;
        tex = that.tex;
        body = that.body;
        return ( B ) this;
    }

    /**
     * Returns whether the builder has enough information to build. For most
     * entities, you need a world and either a Body or an EntityDef.
     *
     * @return boolean
     */
    protected boolean canBuild( ) {
        if ( world == null )
            return false;
        if ( type == null && body == null )
            return false;
        return true;
    }

    /**
     * Returns the reason (if any) the builder does not have enough information
     * to build. Returns empty string if no problems were found.
     *
     * @return String
     */
    protected String whyCantBuild( ) {
        if ( world == null )
            return "World is null.";
        if ( type == null && body == null )
            return "No type/body specified.";
        return "";
    }

    /**
     * Returns an entity created from given data.
     *
     * @return Entity
```

```java
     */
    public Entity build( ) {
        Entity out = null;
        if ( canBuild( ) ) {
            if ( type != null ) {
                out = new Entity( name, type, world, pos, rot, sca, tex, solid );
            } else {
                out = new Entity( name, pos, tex, body, solid );
            }
        }
        prepareEntity( out );
        return out;
    }

    protected void prepareEntity( Entity out ) {
        if ( out != null ) {
            if ( mover != null ) {
                out.addMover( mover, RobotState.IDLE );
            }
            if ( soundlines.size > 0 ) {
                SoundManager soundMan = out.getSoundManager( );
                if ( soundMan == null ) {
                    soundMan = new SoundManager( );
                    out.setSoundManager( soundMan );
                }
                for (String line: soundlines){
                    soundMan.getSoundWithProperties( line );
                }
            }
            out.postLoad( );
        }
    }

    protected static final String nameTag = "Name";
    protected static final String typeTag = "Definition";
    protected static final String xTag = "X";
    protected static final String yTag = "Y";
    protected static final String aTag = "Angle";

}
```